Delphi 6

求名实典

林锐 等编



推荐书目

【责任编辑】苏茜
袁秀珍
【封面设计】孙天昭
何合

Delphi 6 实务经典

本书特色：

- 由浅入深，循序渐进，马上掌握学习重点
- 范例向导，从实际操作中学习，效果加倍
- 超过450个范例，说明各种常用的组件与函数
- One-Tiered、Two-Tiered 数据库应用系统开发
- ADO数据库应用系统开发
- Quick Report 报表制作与设计
- Install Shield 安装程序示范
- Multi-Thread 多线程程序设计
- Delphi VCL 组件制作
- 示范各种文件处理技巧
- 详细说明 Object Pascal 程序设计技巧（附录A、B）
- 以详细的范例说明各种常用的内置函数（附录C、D、E）



本书内附范例光盘

ISBN 7-113-04856-0

ISBN7-113-04856-0/TP·767
定价：92.00 元（含盘）

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE
中国铁道出版社·计算机图书批销部
地址：北京市宣武区右安门西街4号
邮编：100054
网址：http://www.tpbook.net
http://www.tpbook.com
读者购书电话：(010) 63545974 (010) 63550055
读者服务电话：(010) 63560056 (010) 63560048

Delphi 6 实务经典

林金霖　编著

中国铁道出版社
2002·北京

（京）新登字 063 号

北京市版权局著作合同登记号：01-2002-1806 号

版 权 声 明

本书中文繁体字版《Delphi 6 实务经典》由第三波资讯股份有限公司出版(2002)，版权归第三波资讯股份有限公司所有。本书简体中文版由第三波资讯股份有限公司授权中国铁道出版社出版(2002)。专有出版权属中国铁道出版社所有，未经本书原版出版者和本书出版者书面许可，任何单位和个人不得以任何形式或任何手段复制或传播本书的一部分或全部。

图书在版编目（CIP）数据

Delphi 6 实务经典/林金霖编著. —北京：中国铁道出版社，2002.8

ISBN 7-113-04856-0

Ⅰ.D… Ⅱ.林… Ⅲ.DELPHI 语言 程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2002)第 060173 号

书　　名：Delphi 6 实务经典

作　　者：林金霖

出版发行：中国铁道出版社（100054，北京市宣武区右安门西街 8 号）

策划编辑：严晓舟　郭毅鹏

责任编辑：苏　南　袁秀珍

封面设计：孙天昭　何　合

印　　刷：北京兴顺印刷厂

开　　本：787×1092　1/16　印张：61　字数：1476 千

版　　本：2002 年 9 月第 1 版　2002 年 9 月第 1 次印刷

印　　数：1～4000 册

书　　号：ISBN 7-113-04856-0/TP・767

定　　价：92.00 元

版权所有　侵权必究

凡购买铁道版的图书，如有缺页、倒页、脱页者，请与本社计算机图书批销部调换。

出版说明

Delphi 6 是一个功能强大且高效的快速应用程序开发工具，它可协助您快速且容易地开发各种新一代的电子商务应用系统。Delphi 的 BizSnap 运用 Web Service 技术轻松地整合企业与企业(B2B, Business to Business)之间的信息交换。WebSnap 则运用新式软件组件的技术简化网页应用程序的开发复杂度。DataSnap 则是整合各种数据访问的技术以简化多层分布式数据库应用系统的构建。从 Delphi 1.0 到 Delphi 6.0，Borland 公司在每一个版本皆尽了最大的努力来加强 Delphi 的功能。正因为如此，所以软件工程师可以轻松地使用 Delphi 撰写各式各样的应用程序，无论是文件管理、Client/Server 或 Multi-Tier 数据库应用程序、因特网应用程序，甚或是企业间的跨平台的应用程序皆难不倒它。

面对 Delphi 6 这么完整的功能，如何决定本书的章节内容，真是难倒作者。经过几番思索及综合多位读者的意见后，最后决定了本书所显示的内容。至于不足的部分作者会以专门的书籍来讨论数据库及其他高级的内容。本书的重点将着重于基础 Object Pascal 程序撰写的技巧、Delphi 程序结构的彻底认识、各种 VCL 组件的应用、各种文件类型的高级处理、各种不同的数据库应用程序、Client/Server 数据库应用程序及其他 Object Pascal 提高部分的应用，例如：多线程的程序撰写与 Delphi VCL 组件的制作。除此之外，作者另外以四个附录章节来讨论 Object Pascal 与 Delphi 所提供的各种内置函数，希望通过 Object Pascal 的认识与 Delphi 内置函数的应用可以提高您程序开发的效率。

本书由第二波资讯股份有限公司提供版权，经中国铁道出版社计算机图书项目中心审选，王占清、李之明、李自运、崔仙翠、敖省林、陈兰芳等同志完成了本书的整稿及编排工作。

最后，希望本书可以协助您打开面向对象程序设计的大门，如果您对本书有任何的批评与指教，欢迎 E-mail 至 carlos1@ms22.hinet.net。

中国铁道出版社

2002 年 8 月

目 录

第 1 章 认识 Delphi 6 1

1.1 Delphi 6 简介 2
1.2 Delphi 6 集成性开发环境 4
1.2.1 加速栏(Speed Bar) 4
1.2.2 组件面板(Component Palette) 7
1.2.2.1 多页标签组件面板 7
1.2.2.2 组件面板的分类 7
1.2.2.3 如何加入组件 8
1.2.3 对象检查器(Object Inspector) 9
1.2.4 代码编辑器(Code Editor) 12
1.2.5 代码浏览器(Code Explorer) 17
1.2.6 窗体(Form) 18
1.2.7 对象树状视图 19
1.3 鼠标右键功能菜单(Popup Menus) 19
1.4 自定义桌面设置 20
1.5 预定工作表(To-Do List) 20
1.6 如何获取更多的相关信息 21
1.6.1 Delphi 6 使用手册 22
1.6.2 Delphi 6 联机帮助说明 22
1.6.3 Borland Web Site 25
1.6.4 Delphi 深度历险网站 26
1.6.5 加入 Delphi Chart 26

第 2 章 Delphi 6 程序结构 27

2.1 三秒钟完成第一个程序 28
2.2 了解工程(Project) 30
2.2.1 什么是工程(Project) 30
2.2.2 工程文件(.DPR) 30
2.2.3 窗体文件(.DFM) 35
2.2.4 程序单元文件(.PAS) 36
2.3 利用工程管理器(Project Manager) 39
2.4 工程选项设置(Project Options) 41
2.5 利用对象库(Object Repository) 43

2.5.1 使用对象库的对象 43
2.5.2 创建自己的对象库 45

第 3 章 Delphi 6 对象共同属性 49

3.1 与位置或大小有关的属性 51
3.2 与显示或外表有关的属性 53
3.2.1 BorderStyle 属性 53
3.2.2 Color 属性 55
3.2.3 Font 属性 57
3.3 与 Parent 有关的属性 59
3.4 与引导有关的属性 61
3.4.1 Caption 属性 61
3.4.2 TabOrder 与 TabStop 属性 62
3.5 与拖动有关的属性 63
3.5.1 Drag-and-Drop 自动拖动 64
3.5.2 Drag-and-Dock 拖动 67
3.6 设置共同属性 69
3.7 Parent、Owner、Sender 与 Self 70
3.7.1 Parent 与 Owner 71
3.7.2 Sender 72
3.7.3 Self 74

第 4 章 Delphi 6 对象共同事件 75

4.1 使用默认的事件处理过程 76
4.1.1 如何使用默认的事件处理过程 76
4.1.2 如何编写共享的事件处理过程 77
4.2 鼠标事件处理过程 78
4.2.1 Mouse Down 鼠标事件 79
4.2.2 Mouse Move 鼠标事件 80
4.2.3 Mouse Up 鼠标事件 81
4.2.4 鼠标事件综合练习 82
4.2.5 Click 与 DblClick 鼠标事件 89
4.2.6 MouseWheel 鼠标滚轮事件 90
4.2.7 MouseWheel Down 鼠标事件 91
4.2.8 MouseWheel Up 鼠标事件 91
4.3 键盘事件 93
4.3.1 KeyDown 键盘事件 93

4.3.2 KeyUp 键盘事件 96
4.3.3 KeyPress 键盘事件 97

第 5 章 TForm 对象 99

5.1 TForm 对象的属性 100
5.1.1 与活动(Active)相关的属性 102
5.1.2 与图标(Icon)相关的属性 103
5.1.3 与视觉相关的属性 106
5.1.4 与引导相关的属性 111
5.1.5 与绘图相关的属性 111
5.1.6 与菜单相关的属性 111
5.1.7 与键盘相关的属性 114
5.1.8 与滚动条(ScrollBar)相关的属性 115
5.1.9 与控制组件相关的属性 117
5.2 TForm 对象的操作方法 120
5.2.1 TForm 对象取得与释放 121
5.2.2 TForm 对象显示与隐藏 125
5.2.3 TForm 对象产生方式 131
5.2.4 TForm 对象焦点(Focus)设置 132
5.2.5 对象控制的操作方法 132
5.3 TForm 对象的事件处理 133
5.3.1 鼠标相关事件 134
5.3.2 键盘相关事件 134
5.3.3 拖动相关事件 134
5.3.4 创建与释放相关事件 134
5.3.5 其他相关事件 139

第 6 章 Delphi 6 按钮控制对象 145

6.1 TButton 对象 146
6.2 TBitButton 对象 148
6.3 TToolBar 对象 151
6.3.1 TToolBar 对象属性 152
6.3.2 TToolButton 对象属性 156
6.4 TCoolBar 对象 158
6.4.1 TCoolBar 对象属性 158
6.4.2 TCoolBand 对象属性 159
6.5 TUpDown 与 TSpinEdit 对象 161

6.6 TSpeedButton 对象 163
6.7 TSpinButton 对象 167

第 7 章 Delphi 6 文本控制对象 169

7.1 TEdit 对象 170
7.1.1 TEdit 对象的属性 172
7.1.2 TEdit 对象的操作方法 176
7.1.3 TEdit 对象的事件处理 182
7.2 TMaskEdit 对象 183
7.2.1 TMaskEdit 对象属性 183
7.2.2 TMaskEdit 对象掩码字符 185
7.3 TMemo 对象 187
7.4 TRichEdit 对象 193
7.4.1 TRichEdit 对象属性 195
7.4.2 TRichEdit 对象操作方法 198
7.5 TLabel 对象 201
7.6 TStringGrid 对象 205
7.7 TLabeledEdit 对象 208

第 8 章 Delphi 6 功能选择对象 211

8.1 TMainMenu 对象 212
8.1.1 下拉式菜单的制作 212
8.1.2 TMainMenu 对象属性 213
8.1.3 TMenuItem 对象 215
8.1.4 动态创建 TMainMenu 对象菜单 218
8.2 TPopupMenu 对象 219
8.3 TListBox 对象 221
8.3.1 TListBox 对象属性 221
8.3.2 TListBox 对象操作方法 225
8.4 TComboBox 对象 228
8.5 TComboBoxEx 对象 230
8.5.1 TComboBoxEx 对象常用属性 231
8.5.2 TComboExItems 对象 231
8.6 TRadioButton 对象 239
8.7 TCheckBox 对象 240
8.8 TCheckListBox 对象 243
8.9 TTreeView 对象 245

8.9.1 TTreeView 对象常用属性与操作方法 246
8.9.2 TTreeNode 对象 248
8.9.3 TTreeView 对象与数据库 253
8.10 TListView 对象 260

第 9 章 Delphi 6 分类对象 267

9.1 TGroupBox 对象 268
9.2 TRadioGroup 对象 268
9.3 TPanel 对象 272
9.4 TScrollBox 对象 275
9.5 TPageControl 对象 276
9.5.1 TPageControl 对象属性 277
9.5.2 TTabSheet 对象 282
9.6 THeaderControl 对象 289

第 10 章 Delphi 6 常用对话框（一） 295

10.1 ShowMessage 对话框 296
10.2 MessageDlg 对话框 296
10.3 MessageBox 对话框 301
10.4 InputBox 对话框 304
10.5 InputQuery 对话框 305
10.6 OpenDialog 与 SaveDialog 对话框 306
10.7 OpenPictureDialog 与 SavePictureDialog 对话框 310
10.8 FontDialog 与 ColorDialog 对话框 313
10.8.1 FontDialog 常用属性与操作方法 314
10.8.2 ColorDialog 常用属性与操作方法 317
10.9 PrintDialog 与 PrintSetupDialog 对话框 319
10.9.1 PrintDialog 对话框 319
10.9.2 PrintSetupDialog 对话框 321
10.10 FindDialog 与 ReplaceDialog 对话框 323

第 11 章 Delphi 6 常用对话框（二） 329

11.1 ShowMessagePos 对话框 330
11.2 ShowMessageFmt 对话框 331
11.3 MessageDlgPos 对话框 336
11.4 CreateMessageDialog 对话框 338
11.5 LoginDialog 对话框 340

11.6 LoginDialogEx 对话框 341
11.7 RemoteLoginDialog 对话框 343
11.8 SelectDirectory 对话框 345
11.9 PromptForFileName 对话框 348

第 12 章 Delphi 6 文件目录管理对象 351

12.1 TDirveComboBox 对象 352
12.2 TDirectoryListBox 对象 352
12.3 TFileListBox 对象 355
12.4 TFilterComboBox 对象 358
12.5 TShellComboBox 对象 359
12.6 TShellTreeView 对象 362
12.7 TShellListView 对象 364
12.8 TShellChangeNotifier 对象 368

第 13 章 Delphi 6 图像与绘图管理对象 371

13.1 TCanvas 对象 372
13.1.1 TPen 画笔对象 378
13.1.2 TBrush 笔刷对象 380
13.1.3 TFont 字体对象 381
13.2 TShape 对象 382
13.3 TImage 对象 388
13.4 TImageList 对象 390
13.5 TPainBox 对象 391
13.6 TBevel 对象 393
13.7 TScreen 对象 393
13.8 TColorGrid 对象 396
13.9 TColorBox 对象 397
13.10 TAnimate 对象 398
13.11 TMediaPlayer 对象 400

第 14 章 Delphi 6 Action 管理对象 403

14.1 TActionList 对象 404
14.2 自定义 TAction 对象 412
14.3 TActionManager 对象 414
14.4 TActionMainMenu 对象 415
14.5 TActionToolBar 对象 416

14.6 TCustomizeDlg 对象 417

第 15 章 Delphi 6 其他常用对象 419

15.1 TScrollBar 对象 420
15.2 TProgressBar 对象 424
15.3 TStatusBar 对象 426
15.4 TTimer 对象 429
15.5 TSplitter 对象 433
15.6 TDateTimePicker 对象 434
15.7 TMonthCalendar 对象 437
15.8 TFrame 对象 442
15.9 TApplication 对象 445
15.10 TRegistry 对象 450
15.11 TIniFile 对象 453
15.12 TList 对象 457
15.13 TStrings 与 TStringList 对象 468

第 16 章 MDI 应用程序 473

16.1 多文档界面(MDI)应用程序 474
16.2 利用对象库创建 MDI 应用程序 475
16.3 手动创建 MDI 应用程序 481
16.3.1 创建 MDI 应用程序主窗口 481
16.3.2 创建 MDI 子窗口 486
16.3.3 主窗口文件功能设置 490
16.3.4 创建其他的窗口(如 AboutBox 窗口) 492

第 17 章 Delphi 6 数据库程序设计(一) 497

17.1 数据库简介 498
17.2 Delphi 6 与数据库环境设置 498
17.2.1 BDE Administrator 499
17.2.2 Paradox、dBase 及 FoxPro 的连接 500
17.2.3 MS SQL Server 的连接 502
17.2.4 Oracle Server 的连接 511
17.2.5 以 ODBC 连接 Access 数据库 520
17.3 Database Desktop 524
17.3.1 创建新的数据表(Table) 525
17.3.2 为数据表(Table)添加数据 526

17.3.3 查询数据表(Table)的数据 528
17.3.3.1 以 SQL 命令查询数据库 528
17.3.3.2 以 QBE 访问异构数据库 528
17.4 利用向导创建数据库程序 529
17.4.1 向导创建简单的数据库程序 530
17.4.2 向导创建 Master-Detail 的数据库程序 533
17.4.3 手动创建 Master-Detail 的数据库程序 539
17.5 数据库查询语言 SQL (Structured Query Language) 540
17.5.1 SELECT 命令 540
17.5.2 UPDATE 命令 544
17.5.3 DELETE 命令 545
17.5.4 INSERT 命令 545

第 18 章 Delphi 6 数据库程序设计(二) 547

18.1 TDataSet 对象 548
18.1.1 TDataSet 对象状态 548
18.1.2 TDataSet 对象属性 549
18.1.3 TDataSet 对象操作方法 554
18.1.4 TDataSet 对象的事件处理 564
18.2 TTable 对象 573
18.2.1 TTable 对象数据过滤 573
18.2.1.1 利用 Filter 属性进行数据过滤 573
18.2.1.2 利用 onFilterRecord 事件进行数据过滤 575
18.2.2 TTable 对象数据搜索 577
18.2.2.1 Locate 搜索 577
18.2.2.2 LookUp 搜索 579
18.3 TQuery 对象 581
18.3.1 设计时期的 TQuery 对象 581
18.3.2 运行时期的 TQuery 对象 583
18.3.3 动态产生 SQL 语句 584
18.3.4 利用 SQL 语句动态查询(以 Oracle 为例) 587
18.3.5 利用 SQL 语句动态查询(以 MSSQL 为例) 590
18.4 TStoredProc 对象 594
18.4.1 TStoredProc 对象访问 Oracle 数据库 594
18.4.1.1 Oracle 数据库的 Stored Procedure 594
18.4.1.2 Oracle 数据库的 Stored Function 596
18.4.1.3 Oracle 数据库的 Stored Package 599
18.4.2 TStoredProc 对象访问 MS SQL 数据库 602

18.4.2.1 MS SQL 数据库的 Stored Procedure(一) 603
18.4.2.2 MS SQL 数据库的 Stored Procedure(二) 604
18.4.2.3 MS SQL 数据库的 Stored Procedure(三) 606

第 19 章 Delphi 6 数据库程序设计(三) 609

19.1 数据库程序设计常用的对象 610
19.1.1 TDataModule 对象 610
19.1.2 TDatabase 对象 612
19.1.3 TDBGrid 对象 616
19.1.3.1 TDBGrid 对象的常用属性 616
19.1.3.2 TDBGrid 对象的常用事件 621
19.1.4 TDBCtrlGrid 对象 624
19.1.5 TDBNavigator 对象 627
19.2 Delphi 6 的报表对象 628
19.2.1 Delphi 6 报表的概念 629
19.2.2 TQuickRep 对象的常用操作方法 630
19.2.3 直列式(List)报表制作 630
19.2.4 标签式(Label)报表制作 635
19.2.5 分组式(Group)报表制作 638
19.2.6 Master/Detail 报表制作 640
19.3 Delphi 6 的安装程序 643

第 20 章 Delphi 6 与 ADO 组件 653

20.1 何谓 ADO(ActiveX Data Objects)组件 654
20.2 TADOConnection 对象 656
20.2.1 TADOConnection 对象常用属性 656
20.2.2 TADOConnection 对象常用操作方法 667
20.2.3 TADOConnection 对象事件处理 670
20.3 TADODataSet 对象 675
20.3.1 通过 TADODataSet 对象获取数据 675
20.3.2 TADODataSet 对象的批量更新处理 681
20.4 TADOTable 对象 683
20.5 TADOQuery 对象 685
20.6 TADOStoredProc 对象 686
20.7 TADOCommand 对象 688

第 21 章 Delphi 6 多线程程序.....691

21.1 多线程(Multi-Thread)程序简介.....692
21.2 第一个多线程程序.....695
21.3 多线程与文件复制.....704
21.4 多线程与数据库应用程序.....710

第 22 章 编写 Delphi 6 VCL 组件.....713

22.1 如何编写 Delphi VCL 组件.....714
22.2 加在组件面板上的图形文件.....719
22.3 创建组件属性(Property).....722
22.3.1 公开派生类的属性.....722
22.3.2 添加直接访问属性.....723
22.3.3 设置属性默认值.....725
22.3.4 添加间接访问属性.....726
22.3.5 另一种间接访问属性.....727
22.4 创建组件的操作方法(Method).....729
22.5 创建组件的事件(Event).....731
22.6 VCL 组件制作实例.....732
22.6.1 修改既有的 VCL 组件.....732
22.6.2 制作对话框的 VCL 组件.....737
22.6.3 制作访问注册表(Registry)的 VCL 组件.....741

附录 A Object Pascal 程序设计（一）.....747

A.1 Object Pascal 程序结构.....748
A.2 Console 应用程序.....749
A.3 Windows 应用程序.....751
A.4 工程与程序单元文件.....755
A.5 程序注释.....756
A.6 变量声明.....757
A.7 常量声明.....759
A.8 语句(Statement).....759
A.8.1 赋值语句.....760
A.8.2 过程与函数调用语句.....760
A.8.3 Goto 语句.....760
A.8.4 复合语句.....760
A.8.5 条件语句.....761

A.8.6 循环语句 763
A.8.6.1 Repeat 循环语句 763
A.8.6.2 While 循环语句 763
A.8.6.3 For 循环语句 764
A.8.6.4 循环中止语句 765
A.8.7 异常处理语句 766
A.8.7.1 try...except...end 语句 766
A.8.7.2 try...finally...end 语句 768
A.8.7.3 自定异常类 769
A.8.7.4 再次触发异常 770
A.9 过程(Procedure)与函数(Function) 770
A.9.1 过程(Procedure)的定义与声明 771
A.9.2 函数(Function)的定义与声明 773
A.9.3 参数的传递 774
A.9.3.1 传值参数 774
A.9.3.2 变量参数 775
A.9.3.3 常量参数 776
A.9.3.4 输出参数 776
A.9.3.5 参数默认值 777
A.9.3.6 数组参数 777
A.9.4 过程与函数的重载(Overloading) 778
A.10 程序块 779

附录 B Object Pascal 程序设计（二） 783

B.1 表达式 784
B.2 运算符 785
B.2.1 算术运算符(Arithmetic Operators) 785
B.2.2 布尔运算符(Boolean Operators) 787
B.2.3 逻辑运算符(Logical Operators) 788
B.2.4 字符串运算符(String Operators) 790
B.2.5 集合运算符(Set Operators) 791
B.2.6 关系运算符(Relational Operators) 792
B.2.7 类运算符(Class Operators) 792
B.3 Simple Types(简单类型) 793
B.3.1 Integer Types(整数数据类型) 796
B.3.2 Character Types(字符类型) 797
B.3.3 Boolean Types(布尔类型) 798
B.3.4 Enumerated Types(枚举类型) 799

B.3.5 SubRange Types(子界类型) 801
B.3.6 Real Types(实数类型) 801
B.4 String Types(字符串类型) 802
B.5 Structured Types(结构类型) 803
B.5.1 Set Types(集合类型) 803
B.5.2 Array Types(数组类型) 803
B.5.3 Record Types(记录类型) 805
B.6 Pointer Types(指针类型) 808
B.7 Variant Types(变体类型) 809

附录 C Delphi 6 常用文件处理函数 811

C.1 目录处理函数 812
C.2 文件名处理函数 817
C.3 文件属性处理函数 823
C.4 文件处理函数 826
C.5 文本文件处理函数 833
C.6 类型文件(Typed File)处理函数 841
C.7 无类型文件(Untyped File)处理函数 847

附录 D Delphi 6 常用日期与时间函数 853

D.1 获取特定的日期与时间 854
D.2 日期处理函数 857
D.3 时间处理函数 872
D.4 比较与判断函数 882
D.5 日期与时间的编译与解译 893
D.6 日期与时间的转换函数 900

附录 E Delphi 6 其他常用函数 905

E.1 Pascal 字符串处理函数 906
E.2 以 Null 结尾的字符串处理函数 930
E.3 数值处理函数 947
E.4 Variant 处理函数 954

第1章

认识 Delphi 6

本章目标

本章主要说明 Delphi 6 的新功能，阅读本章可以让您对 Delphi 6 有更进一步的认识，了解您正在学习什么样的开发工具。Delphi 6 是一个全方位的开发工具，无论是数据库或因特网应用程序，Delphi 6 皆可以轻易地完成它。本章除了说明 Delphi 6 的新功能外，更详细地说明了有关 Delphi 6 集成性开发窗口的使用方法，最后告诉大家如何获取更多的帮助说明，希望大家能善用现有资源，加速您的学习效果。

本章主题

1.1 Delphi 6 简介

1.2 介绍 Delphi 6 集成性开发环境

1.3 鼠标右键功能菜单

1.4 自定义桌面设置

1.5 预定工作表(To-Do List)

1.6 如何取得更多的相关信息

本章内容主要分成六个小节：

- 在“Delphi 6 简介”小节中，我们将简要地介绍 Delphi 6 的轮廓及特性，让读者可以快速地进入面向对象程序设计的天堂。
- 在“Delphi 6 集成性开发环境”小节中，主要针对 Delphi 6 集成性开发环境进行介绍，并介绍 Delphi 6 开发工具的使用，以协助读者能更快速且简便地完成工作。
- 在“鼠标右键功能菜单”小节中，将介绍鼠标右键在每个不同区域的功能。
- 在“自定义桌面设置”小节中，将介绍如何在 Delphi 6 的 IDE 环境中建立自定义的桌面。
- 在“预定工作表(To-Do List)”小节中，将介绍如何使用 Delphi 6 的 To-Do List 功能。
- 在“如何获取更多的相关信息”小节中，将告诉读者何处可以获取更多的相关信息及帮助。

1.1 *Delphi 6* 简介

Borland Delphi 6 是由 Borland 公司所发表的 e 时代的开发工具，它也是企业级可视化面向对象快速开发工具(RAD，Rapid Application Development)。目前 Delphi 6 的产品包括企业版(Enterprise)、专业版(Professional)和个人版(Personal)三种版本。本书将以 Enterprise 版本作为介绍环境。

Delphi 6 有相当多的功能，以下将对其作简略地介绍，如果您觉得很多名词没有看过或听过，没有关系，不要灰心；如果以下的名词您都知道，那我想您也没有看本书的必要了。所以啦！以下这一段您就以非常轻松的态度来阅读，以不睡觉为原则喔！其主要功能有以下几点(摘自 Borland 公开发行刊物)：

1. BizSnap 运用 Web Service 技术以简化企业与企业之间信息交换集成的复杂度
 Web Service 技术是以 XML、SOAP 及 HTTP 等业界标准为基础，以软件组件的方式通过因特网提供服务与数据。这项 Web Service 技术获得 IBM、Microsoft、Sun Microsystems、HP 及 Oracle 等厂商共同支持。另外，Microsoft 已经将这项技术运用于最新的软件工具产品——.NET 上。因为 BizSnap 是运用 Web Service 技术，因此可以轻松地将现有的企业信息转换为 XML 文件，以达到企业间的信息集成。BizSnap 技术让 Delphi 的应用程序能够通过标准的 SOAP/XML Web Service 与 Microsoft .NET、Microsoft BizTalk 以及 Sun Microsystems ONE 等应用程序相集成。
2. DataSnap 技术可构建高性能的多层分布式数据库应用系统
 Delphi 6 的 DataSnap 技术可让您通过 SOAP、CORBA、COM 及 TCP/IP 等分布式运算标准，开发多层分布式数据应用程序。搭配 DataSnap 所开发出的中间软件(Middleware)，客户端(Client)的应用程序无需任何配置设置，大大地降低客户端的维护成本。DataSnap 中间软件的各项技术能对数据库的连接(Connection)进行有效地管理，以达到带宽的有效控制。DataSnap 技术再搭配上 Web Service 更能够流畅地集成企业与上下流厂商间的数据交换，可让您所编写的应用系统扩展成 e-Business 应用系统。

3. WebSnap 技术提高 Web 应用程序的开发速度
 Delphi 6 的 WebSnap 技术让您在开发 Web 应用程序时享有 RAD 工具独有的便利性。WebSnap 的调试服务器更是无需借助额外的 Web 服务器，即可在 IDE 集成开发环境内使用 Delphi 所提供的 Web 应用程序完整的调试功能。除此之外，您可以使用 Microsoft FrontPage 或 Macromedia Dreamweaver 等知名的网页制作工具先行制作网页画面及编写服务端的脚本(Script)语言，再通过 WebSnap 的相关组件与 VBScript、JScript 等前台的脚本语言直接集成。通过这样的集成，您可以同时获得脚本语言的灵活性与 Delphi 的执行效率。Delphi 另外提供 WebPage 外观设计向导，以可视化工具与软件组件结合的方式，快速开发集成数据库及动态网页的 Web 应用程序。
4. CLX 跨平台组件库可开发出适用于 Windows 及 Linux 平台的应用程序
 Linux 操作系统是一个具高可靠性、高扩展性的免费操作系统，Linux 操作系统所提供的网页服务器已经是一个经过市场验证的成熟解决方案。在一般的应用程序方面，Linux 也快速成为 Microsoft Windows 操作系统的替代方案。Kylix 是 Borland 在 Linux 操作系统中所提供的快速开发工具，为了让代码能够与 Kylix 完全兼容，Delphi 6 提供一组与 Kylix 完全相同的 CLX 跨平台组件库。您只要将 Delphi 6 以 CLX 所编写的代码，再利用 Kylix 重新编译即可在 Linux 操作系统上执行。
5. Delphi 6 内置多种数据库访问方式以访问企业的各种信息
 Delphi 6 内置多种数据库访问方式，例如可跨 Linux 操作系统的 dbExpress 驱动程序，它提供高速访问 Oracle、IBM DB/2、Borland InterBase 及 MySQL 等数据库的能力。另外，新版的 BDE(Borland Database Engine)提供高性能的 SQL Link 驱动程序用来访问 Oracle、IBM DB/2、Microsoft SQL Server、Sybase、Informix 及 Borland InterBase 等大型数据库。除此之外，DBE 针对 Oracle 对象关系扩展功能提供完整的支持。dbGO 系统组件则是通过 Microsoft ADO 来访问各类的数据源。无论您是使用哪一种数据库访问机制，都可配合 Delphi 6 所提供的数据感知组件(Data aware component)快速开发出数据库应用程序或 Web 应用程序。
6. 运用 CORBA 集成异构环境
 CORBA 是由 800 余家 IT 厂商参与的 OMG(Object Management Group)组织所制定的分布式对象规格，也是众多厂商解决异构环境的一个重要桥梁。运用您现有的 Delphi 经验，不需要学习编写 CORBAIDL，即能快速建立可供多种异构平台下 C++、Java、Smalltalk、COBOL 等程序语言使用的 CORBA 应用对象。
7. 轻便的 XML 个人数据库引擎——MyBase
 Delphi 6 内置 MyBase 个人用途的 XML 数据库引擎，提供在内存中高性能的数据表及具有灵活性的 XML 数据交换格式。MyBase 除了安装与管理都非常容易外，也同时支持 Master/Detail 的关系模式，并支持所有常用的数据类型及近似于 ANSI-92 的 SQL 语法。
8. 新式 Windows 2000 用户接口组件提高开发生产力
 Delphi 6 提供 Windows 2000、Windows Me 及 Office 2000 最新的用户接口组件，以及运行时期能让用户自定义功能的 ActionBand 系统组件。

Delphi 6 除了提供简单易用的可视化开发环境外，更将运行性能快速提高直逼近 C++，

以提供企业信息部门构建主从式应用及分布式多层应用系统。换言之，您可在 Windows 98、Windows Me 及 Windows 2000 等操作平台，使用 Delphi 6 开发任何您想要完成的高效率应用程序，包含一般性质的应用程序、数据库应用程序、因特网相关应用程序等。

1.2 Delphi 6 集成性开发环境

对 Delphi 6 的新功能有了初步的认识之后，您是不是迫不及待地想要对 Delphi 6 进行更进一步的了解呢？是的，本小节主要介绍 Delphi 6 集成性的开发环境，为了方便您程序开发的进行，您至少有两种方法来运行 Delphi 6：

- 在桌面上双击 Delphi 6 的图标(假如您已经在桌面上创建快捷方式)。
- 在“开始 / 程序 / Borland Delphi 6”中运行“Delphi 6”。

当您启动了 Delphi 6 之后，您会看到一个全新的集成开发环境，Delphi 6 提供了一个功能强大且操作便利的对象程序发展环境，其主要包含有加速栏(Speed Bar)、组件面板(Component palette)、对象检查器(Object Inspector)、代码浏览器(Code Explorer)、代码编辑器(Code Editor)、窗体(Form)及对象树状视图(Object Tree View)等，整个外观如图 1.1 所示。

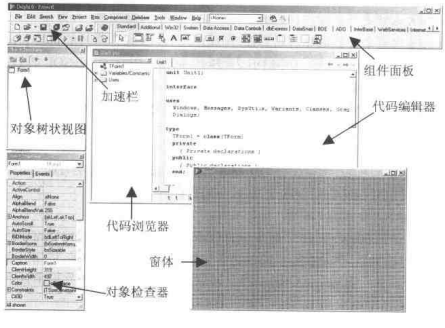


图 1.1 Delphi 6 集成开发环境

下面将针对 Delphi 6 集成开发环境中一些较常用的工具，进行更进一步的介绍。

1.2.1 加速栏(Speed Bar)

加速栏是为了节省进入菜单的时间而设置的功能，Delphi 6 将一些比较常用的功能选项以按钮的方式在加速栏展现，让您可以很快地选取所需的功能，图 1.2 为每个按钮的功能加以说明。

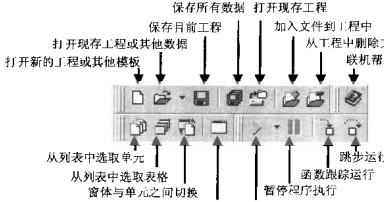


图 1.2　Delphi 6 加速栏选项说明

整个加速栏是由四个 Toolbar 所组成的，包括 Standard Toolbar、View Toolbar、Debug Toolbar 及 Custom Bar。Delphi 6 提供用户自行排列加速栏的功能，您只要使用鼠标左键按住 Toolbar 的最左端，再进行拖动即可，如图 1.3 所示。如果您想设置自己所想要的加速栏，只要在 Toolbar 上按右键，即可选择您所要的 Toolbar 了，如图 1.4 所示。



图 1.3　Delphi 6 加速栏的组成部分

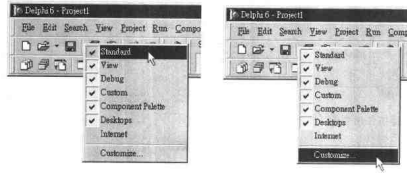


图 1.4　设置加速栏的组成部分　　图 1.5　自定义加速栏

如果您想要自定义加速栏的功能，只要在加速栏上按右键，选择“Customize...”的选项，如图 1.5 所示，即可进行自定义快速功能选项。以下就示范如何在 Standard Toolbar 中加入一个 New Data Module 的功能项，如图 1.6 所示。

(1) 用鼠标左键点选所要添加的项目(Data Module)，会出现 符号。

步骤 2 将鼠标指到所要存放的 Toolbar 上(Standard Toolbar)，并且以左键点选，此时便将您所添加的项目加到指定的 Toolbar 中。

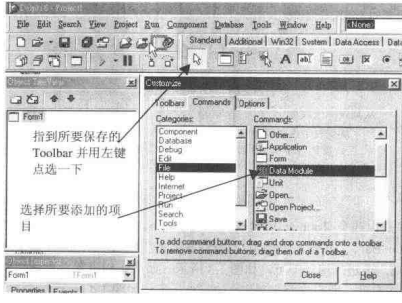


图 1.6 加入一个功能选项到 Toolbar 中

同样的，您也可以将某个功能选项从 Toolbar 中删除，在进行删除之前您必须先在加返栏上按右键，选择“Customize...”的选项(见图 1.5)，并进行下列步骤，请参考图 1.7。

步骤 1 在 Toolbar 中用左键按住所要删除的项目，会出现 这个符号。

步骤 2 移动鼠标至 Customize 的窗口范围并放开左键，删除工作即完成。

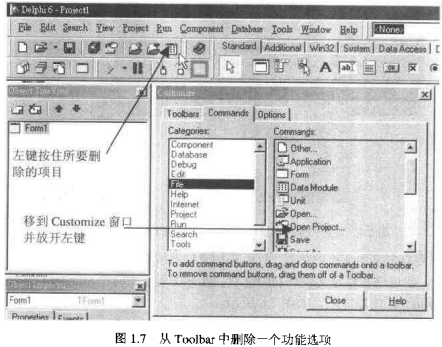


图 1.7 从 Toolbar 中删除一个功能选项

1.2.2　组件面板(Component Palette)

组件(Component)是构成 Delphi 6 应用程序的重要单元，在图 1.1 所提到的窗体(Form)也是组件的一种，Delphi 6 定义了自己的组件标准格式，称之为 VCL(Visual Component Library)。

1.2.2.1　多页标签组件面板

这些组件都放置在组件面板上，由于有太多的组件，所以 Delphi 6 将这些组件分类，给每一类组件一个页标签放置，用户只要点选页标签就可以看见每一类的组件，称之为“多页标签的组件面板”，如图 1.8 所示。



图 1.8　Delphi 6 多页标签的组件面板

1.2.2.2　组件面板的分类

Delphi 6 在组件面板上共提供了 19 种组件类型(以 Enterprise 版为例)，组件可分为可见(visual)和不可见(Non-visual)两种。见表 1.1。

可见组件：是指组件在运行时期(Run Time)与设计时期(Design Time)所看到的外观是相同的，如 Label、Button。

不可见组件：则是在设计时期仅以一个图标(Icon)代表该组件，供程序人员访问或修改，但在运行时期则是无法看见的，例如 Table、DataSource、MainMenu 或 PopupMenu 均为不可见组件。

表 1.1　分别说明各类别的名称

页标签名称	说明
Standard	标准的组件面板
Additional	附加的组件面板
Win 32	Win32 相关的组件面板
System	系统相关的组件面板
Data Access	数据访问组件的组件面板
Data Controls	数据控制组件的组件面板
dbExpress	一组可跨 Linux 平台的数据库访问的组件
DataSnap	一组允许您运算 SOAP、CORBA、COM 及 TCP/IP 等多层分布式数据库应用程序相关组件的组件

续上表

页标签名称	说明
BDE	一组以新版 BDE 为基础的数据库访问组件
ADO	一组完全支持 Microsoft Universal Data Access 的 ADO 组件
InterBase	一组 InterBase 数据库访问组件
WebServices	一组以 XML、SOAP 及 HTTP 为基础的 Web Service 组件
InternetExpress	一组 InterExpress 技术的组件
Internet	开发因特网应用程序的组件
WebSnap	一组可提高 Web 应用程序开发效率的相关组件
FastNet	网络相关组件
Decision Cube	决策立体图表相关组件
QReport	报表相关组件
Dialogs	对话框相关组件
Win31	Windows 3.1 相关组件
Samples	相关的范例组件
ActiveX	ActiveX 相关组件
COM+	COM+相关组件
Indy Clients	另一组网络客户端相关组件
Indy Servers	另一组网络服务端相关组件
Servers	一组与 Office 有关的组件

在看完上面组件分类后，您是不是觉得怎么会有这么多好用的东西呢？这正是 Delphi 6 号称可视化快速工具的理由，其最大的意义便是程序设计师不需要编写太多代码，就能完成许多复杂的应用程序。当您在窗体中加入某个 VCL 组件时，Delphi 6 同时会在窗体内自动产生相对应的源代码，这种尤如 D.I.Y.(do-it-yourself)的程序建立方式，将大幅度提高程序的开发效率，以实现软件 IC 的理想。

1.2.2.3 如何加入组件

以下介绍四种将 VCL 组件加入窗体的方法：

1. 使用鼠标左键在某个特定的组件上双击，Delphi 6 便会自动地将该组件加到窗体的正中央。
2. 使用鼠标左键点选组件面板上的某个组件图标(此时该组件图标会呈凹下状)，再把鼠标移到窗体上(Form)并用左键点一下或以左键画出指定范围，此时 Delphi 6 便会自动地将该组件加到鼠标所在位置。
3. 在点选组件前，先按着【Shift】键不放再按该组件，此时该组件会呈现凹下状且四周出现一个蓝色的外框，此时您可以放开【Shift】键，并且可以在窗体上重复点选以产生多个相同的组件。若您要取消这个功能，只要去点选组件面板上最左边的“箭头”按钮，如图 1.9 所示。

4. 点选菜单栏中的“View/Component List”打开“Components”窗口，在该窗口中选取所要的组件，并按下“Add to form”键即可将该组件加入窗体中，如图 1.10 所示。

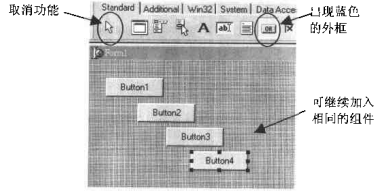


图 1.9　重复加入相同的组件

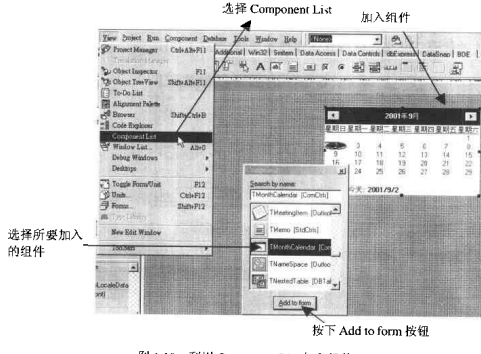


图 1.10　利用 Component List 加入组件



如果您要删除多余的组件，只要点选在窗体上的组件并按下【Del】键即可。

1.2.3　对象检查器(Object Inspector)

对象检查器主要是用来查看或设置窗体上的组件属性(Properties)或组件的事件(Events)。如果对象检查器被其他的窗口盖掉了，您可以按【F11】功能键或点选菜单栏中的“View/Object Inspector”来调出对象检查器，或是直接在对象检查器上方点选鼠标右键菜单的“Stay on Top”选项，如图 1.11 所示。



图 1.11　设置对象检查器为 Stay on top

对象检查器包含三个重要的部分：对象菜单、属性页标签及事件页标签。分别说明如下(参见图 1.12)：

- 对象菜单：对象菜单主要是方便用户可以快速地选取存在窗体上的某个对象。
- 属性页标签：用户可以通过属性页标签设置该对象在设计时期(Design-Time Properties)的各项属性。
- 事件页标签：用户只要在某事件上双击(Double Click)，Delphi 6 便会自动填入该事件的代码，并将光标移到该事件处理(参见图 1.13)。

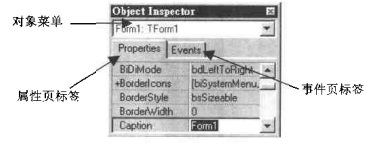


图 1.12　对象检查器



图 1.13　设置对象的事件



一般来说，对象的属性可分为设计时期与运行时期。在对象检查器属性页标签中，所能查看的仅是设计时期所能使用的属性，您可以通过联机帮助文件找出对象所有的属性，步骤如下。

1. 点选所指定的对象(这里以 TForm 对象为例)，并按下【F1】功能键。
2. 此时系统会自行打开联机帮助文件，并以鼠标左键点选“Properties”，此时系统会显示出 TForm 对象的所有属性，您只要进行比较就可以知道哪些是设计时的属性。

在图 1.14 中有■图标者即为设计时期属性，其余则是运行时期属性。

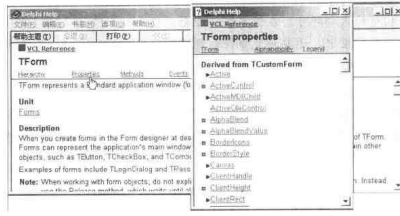


图 1.14　查询对象非设计时期属性

名词解释　对象(Object)：本书所指的对象与一般“面向对象程序设计”中所指的对象是相同的。什么是对象呢?简单地说，可以把对象想象成现实生活中的任何一个物品，人都可以称为一个对象。本质上而言，一个对象包含有三个特点：属性(Property)、方法(Method)及事件(Event)。

名词解释　属性(Property)：以人而言，属性可以称为特征，例如姓名、性别、肤色、身高等。以 Delphi 6 的对象而言，其可能包括 Name(标识名称)、Caption(文本标识)、Left(水平位置)及 Top(垂直位置)等。

名词解释　方法(Method)：对象本身所含的过程或函数即称为方法。通常我们可以通过对象所提供的方法来操作该对象或获取其内部的私有数据。例如我们要关闭一个窗体(Form)，我们可以使用 Close 方法来关闭它，我们完全不需要了解计算机在关闭一个 Form 需要哪些操作。以人而言，就像用手去拿杯子，只要手动就可以了，我们完全不需要知道手为什么会动。

名词解释　事件(Event)：事件就像一个对象的反应行为一般，例如眼睛跑进异物，会流眼泪一样，眼睛跑进异物，触发了“眼睛跑进异物”事件，而在这一个事件中，所定义的内容就是流眼泪。由于 Windows95/98/NT 操作系统是属于“事件驱动”的架构，是有别于传统的“过程性程序语言”一行一行运行的结构，所以在事件驱动的世界中，必须有“事件”发生，程序才会运行；相反的，如果没有事件发生，则整个程序会一直处于停滞状态。一般来说，事件发生的原因，可分为三类：

1. 用户事件：由用户发生的事件。例如用户通过键盘输入、按下鼠标的 Click、Double Click，甚至是移动鼠标时，所触发的相对事件都称为用户事件。
2. 间接事件：用户事件属于直接事件，间接事件则是指由代码触发的事件，例如运行 Show 方法时，会触发 OnShow 的事件。
3. 系统事件：例如系统发出的定时信号所触发的事件。

名词解释

事件处理(Event Handler)：事件处理是指被触发的事件所作出反应的代码。因此您若希望某个对象在某个事件发生后能作出预期的反应，只需在对象的事件处理中编写相对的代码即可。在编写事件驱动的程序时，除了必须注意代码是否放在正确的事件处理中，还要避免事件处理过程又调用事件本身，否则系统会产生无穷循环，那么程序永远也运行不完了。

对象检查器新增对象关系的功能，假如某个组件的属性值为另一个组件，那么就可以直接从对象检查器中查看与设置了组件的相关属性值。例如图 1.15 的 TForm 对象的 Menu 属性值为 TMainMenu 对象，那么您就可以直接从 TForm 对象的 Menu 属性查看 TMainMenu 对象的属性值。



图 1.15　对象检查器的对象关系功能

1.2.4　代码编辑器(Code Editor)

Delphi 6 将代码编辑器(Code Editor)与代码浏览器(Code Explorer)结合在一起，如图 1.16 所示，您当然可以根据个人的爱好重新排列。我们先介绍代码编辑器，稍后再介绍代码浏览器。代码编辑器本身是一个功能完整的程序编辑器，其编辑方式类似于 Word、WordPad。代码编辑器并提供一组编辑功能键(详见表 1.2)，可通过这些编辑功能键来增加程序的编写效率，因为当您利用功能键来编辑程序会快于使用鼠标。代码编辑器可同时编辑多个源文件，并将关键字(Key Word)以粗体来表示，同时也通过不同的字体颜色来区别各个不同意义的程序块。见表 1.2。

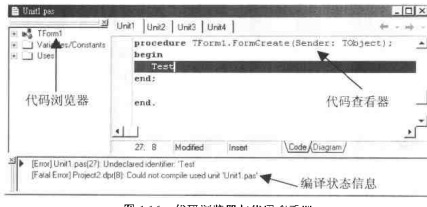


图 1.16　代码浏览器与代码查看器

表 1.2　编辑器常用功能键说明表

功能键	意义
Home	将光标移到每一行的最前端
End	将光标移到每一行的最后端
Ctrl + Home	将光标移到程序的开头
Ctrl + End	将光标移到程序的结尾
Insert	插入/取代模式切换
Del	删除光标后面的字符或是选中的代码
←(退格键)	删除光标前面的字符
Shift+方向键	将代码加以选择
Ctrl + C	复制选择的代码到系统的剪贴板
Ctrl + V	将系统的剪贴板内的数据复制到编辑器
Ctrl + X	复制标识区块的代码到系统的剪贴板，并删除原先选择的代码
Ctrl +K+0,1....9	设置书签
Ctrl +Q+0,1....9	将光标移到书签处
Ctrl + F	查找字符串
Ctrl + R	查找并替换字符串

Delphi 6 代码编辑器提供 Code Insight 的功能，此 Code Insight 功能可分为四个子功能，其中包括 Code completion、Code templates、Code parameters 及 Tool-tip expression evaluation 等，分别说明如下：

1. Code completion

当您编写程序时，可能会声明许多变量、函数、过程或对象等。如果您在声明这些数据同时并没有将它们输出，当您想要在程序中使用它们，可就非常麻烦了，因为您必须回到最初声明的地方开始找起，如果只是几十个声明数据也就算了，那如果上百个、上千个，请问您如何找起呢?Delphi 6 针对这点，特别提供这项功能，只要您按下【Ctrl + Space】键就可以启动 Code completion 窗口，此窗口会将目前所在单元(Unit)内声明的所有变量、函

数、过程或对象等名称列出，帮助您快速获取所要的数据。您只要通过三个步骤即可完成这项工作：

1 按下【Ctrl + Space】键，弹出“Code completion”窗口。或是输入对象名称，并在对象名称后面键入句号，在 0.5 秒后，便会自动弹出“Code completion”窗口。

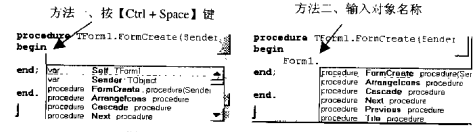


图 1.17 调用 Code completion 窗口的方法

2 利用方向键或鼠标选择所要的项目。

3 选定项目后，再按下空格键或【Enter】键即可。

注意

在您的 Windows 98 系统中，可以正常使用【Ctrl + Space】键吗？如果无法正常使用，把“控制面板/输入法”中的“打开/关闭输入法”组合键“左键”取消(参见图 1.18)。因为 Delphi 6 是外国人设计的，所以他们并不知道【Ctrl + Space】键是我们中文输入法的“打开/关闭”功能。那您可能会问“那以后我的打开／关闭输入法不就不能按了吗”。是的，左边的【Ctrl + Space】键不能用，但是您还是可以使用右边的【Ctrl + Space】键啊！

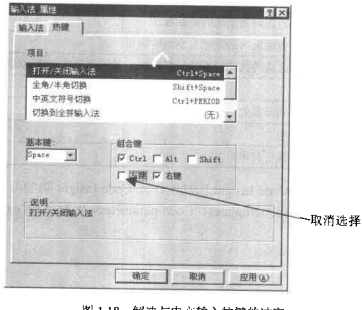


图 1.18 解决与中文输入按键的冲突

2. Code templates

在 Object Pascal 有相当多的语法，因此程序设计师并不见得每一个都记得，因此 Delphi 6

提供 Code templates 的功能，让程序设计师不用记那么多的语法，就能轻松地编写程序。同样的，您只要利用四个简单的步骤便可轻松完成。

1. 在您需要帮助时按下【Ctrl + J】键，弹出命令帮助小窗口。
2. 选择所要的格式。
3. 按下【Enter】键。
4. 填写适当的代码(请参考附录 A、Object Pascal 语法介绍)。

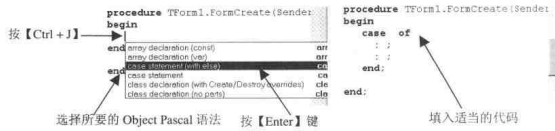


图 1.19　编辑器 Code templates 功能

3. Code parameters

Delphi 6 提供了很多好用的函数，但是程序设计师并无法一一记下来，因此传统的作法可能是翻阅程序开发手册或直接查询 On Line Help。Delphi 6 为了让程序设计成为一件没有负担的乐趣，因此特别提供 Code Parameters 的功能，只要用户在对象的方法或函数名称后输入“左括号”，系统便自动启动 Code parameters 的功能，让用户可以很清楚地知道应该填入哪些参数(参见图 1.20)。

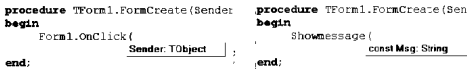


图 1.20　编辑器 Code parameters 功能

4. Tool-tip expression evaluation

这项功能主要用于协助程序调试(Debug)，调试是一件非常伤脑筋的事，Delphi 6 特别提供 Tool-tip expression evaluation 功能来协助程序设计师，以下就两方面来讨论它的强大功能。

- 设计时期的 Tool-tip expression evaluation：在设计时期 Tool-tip expression evaluation 可以让程序设计师很快知道鼠标目前所指的变量或对象是在哪里声明的，例如我们在 Form.onCrate 函数的外面定义一个变量名 MyString，其数据类型为 String。而我们在 Form.onCreate 函数中将此字符串以信息框(MessageBox)的方式加以显示出来。如果我们想要在设计时期知道 MyString 这个变量到底在哪里定义的，我们只要将鼠标指到 MyString 的上面，Delphi 6 会以 Hint 的方式告诉我们，这个变量在 Unit1.pas 的第 19 行定义的(参见图 1.21)。



图 1.21　设计时期的 Tool-tip expression evaluation

- 运行时期的 Tool-tip expression evaluation：在设计时期 Tool-tip expression evaluation 可以让程序设计师很快知道鼠标目前所指的变量或对象的内容是什么。延续上面的例子，如果我们在运行时期(调试时期)想要知道变量 MyString 的内容到底是什么，只要将鼠标指到 MyString 的上面，Delphi 6 会以 Hint 的方式告诉我们，这个变量的内容是什么(参见图 1.22)。

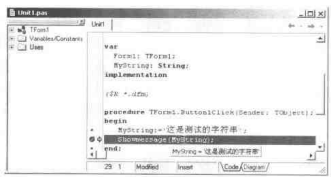


图 1.22　运行时期的 Tool-tip expression evaluation

注意

您可自行决定 Code Insight 要提供哪些功能及其相关的显示时间。如果您要设置，请选择菜单栏上的“Tools/ Editor Options”，将会出现 Editor Properties 窗口，再选择“Code Insight”页标签，即可进行设置(参见图 1.23)。

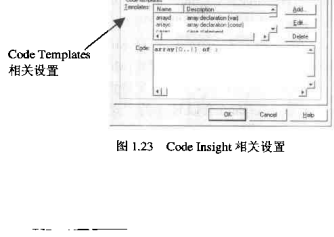


图 1.23　Code Insight 相关设置



代码编辑器(Code Editor)另外提供一个近似于因特网浏览器的功能。例如您想要了解某个单元(Unit)的内容，或是某个组件最初的定义是什么，您只要按住【Ctrl】键不放，并将鼠标指到该单元名称或组件名称上方，按下鼠标左键即可得到更进一步的详细内容。例如在窗体 Form1 中加入一个按钮(Button)，我们想要进一步了解 TButton 的内容是什么，只要按住【Ctrl】键不放，并在 TButton 的上方按下鼠标左键即可。若想回到原来的地方，只要按下右上方的【BackSpace】键即可(参见图 1.24)。

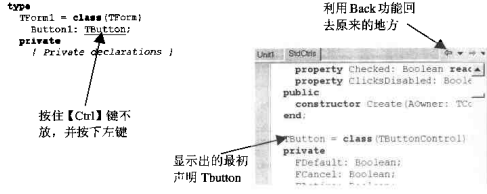


图 1.24　浏览代码

1.2.5　代码浏览器(Code Explorer)

Delphi 6 提供一个层级树状式的方式让您管理代码。以下用一个例子来说明代码浏览器的架构。例如，我们在窗体 Form1 加入三个按钮，所以在图 1.25 中，您便可清楚地看到有三个按钮位于窗体 Form1 下方，分别是 Button1、Button2 及 Button3(参见图 1.25)。

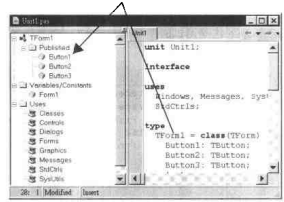


图 1.25　层级树状式的代码管理

在代码浏览器中提供一个快速的查找功能，您可以利用【Ctrl + Shift + ↑】键或【Ctrl + Shift + ↓】键快速地在函数的声明与函数的内容间快速移动，但您必须同时按下【Ctrl + Shift】键，否则系统会自动打开中文输入法或切换中文输入法。以下用一个例子来说明如何在代码浏览

器中使用快速查找功能。

(1) 在窗体(Form)中加入三个按钮(TButton)。

(2) 点选 Button1，在对象检查器的“事件页标签”中，找到 onClick 的事件，双击鼠标左键 (参见图 1.26)。

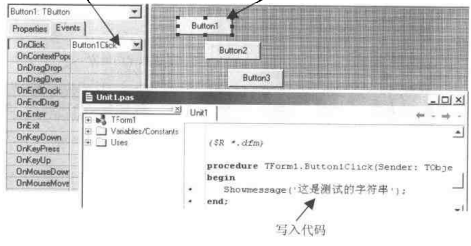


图 1.26 在 Button 的 onClick 中填入代码

(3) 此时在代码浏览器中的 Button1 下方出现一个 ButtonClick 的事件函数名称。在 ButtonClick 的事件代码中，填入“ShowMessage('这是测试的字符串');”的代码。

(4) 此时可利用【Ctrl + Shift + ↑】键或【Ctrl + Shift + ↓】键在函数的声明与函数的内容间快速移动(参见图 1.27)。

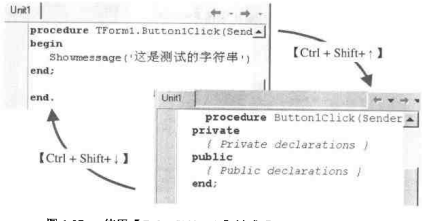


图 1.27 使用【Ctrl + Shift + ↑】键或【Ctrl + Shift + ↓】键快速地在函数的声明与函数的内容间切换

1.2.6 窗体(Form)

窗体(Form)是 Delphi 6 用来放置 VCL(Visual Component Library)组件的容器(container)，

所有可见及不可见的组件都可以放置在窗体内。这里所谓的放置就好像一部车子可以安装很多不同的零件，例如轮胎、玻璃、方向灯等。车体本身是一个组件，而轮胎、玻璃、方向灯也是一个组件，但是它们却可以同时包含在车体内，组合成一个新的大组件，有关窗体(Form)的进一步介绍，请参考第 5 章。

1.2.7　对象树状视图

对象树状视图 (Object Tree View)可协助您更方便地选取程序中所使用的任何组件，如图 1.28 所示。

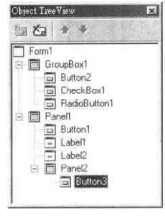


图 1.28　Object Tree View

1.3　鼠标右键功能菜单(Popup Menus)

图形化用户接口(GUI，Graphical User Interface)带给用户很多的便利，图形化用户接口除了提供视觉上的改善外，另外一个突破便是 Popup Menus。Delphi 6 也提供这个功能，在 Delphi 6 的 IDE 环境下，任何地方均有提供鼠标右键菜单的功能，这种设计上要目的与快速功能的按钮有类似的功能，可方便程序的设计。Delphi 6 IDE 环境中常用的鼠标右键功能菜单(Popup Menus)包括对象检查器、快速功能键、组件面板、代码编辑器、窗体对象及代码浏览器等，如图 1.29、图 1.30 所示。



对象检查器　　快速功能键　　组件面板

图 1.29　Delphi 6 IDE 环境右键菜单之

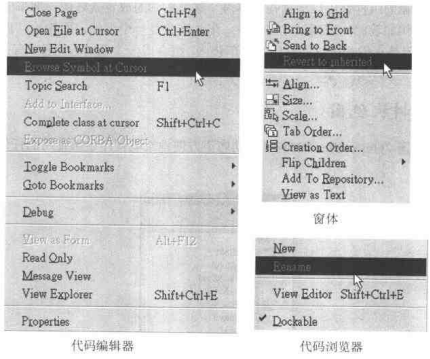


图 1.30 Delphi 6 IDE 环境右键菜单之 一

1.4 自定义桌面设置

从 Delphi 6 开始提供您自定义及保存桌面的设置。在 Delphi 6 的 IDE 环境中有一个 Desktop 工具栏，它包含已保存的桌面设置列表及两个可供自定义桌面的图标，如图 1.31 所示。您可以设置多个桌面，并且能快速在多个桌面间切换。

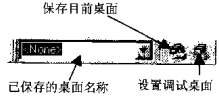


图 1.31 自定义桌面

1.5 预定工作表(To-Do List)

Delphi 6 的预定工作表(To-Do List)记录着工程内预计要进行的任务，您可以直接在事件处理过程中按右键，并选择“Add To-Do Item”以加入一个新的预定工作项目，如图 1.32 所示。

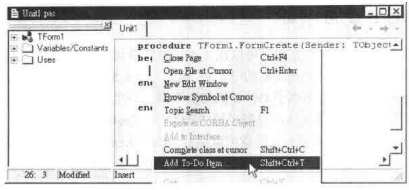


图 1.32　添加预定工作项目

在每一个预定工作项目中，您必须填写 Text(预定工作项目名称)、Priority(优先权)、Owner(所有人)及 Category(分类名称)等字段，以供日后或其他人查询，如图 1.33 所示。

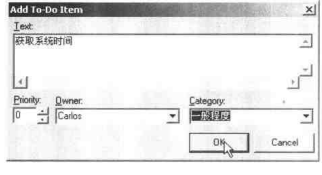


图 1.33　设置预定工作项目内容

如果您想要新增或查看与整个工程有关的预定工作项目，您可以选择菜单栏中的“View/To-Do List”以打开 To-Do List 对话框。假如某个工作项目被画上删除线，则表示该工作项目已经完成，如图 1.34 所示。

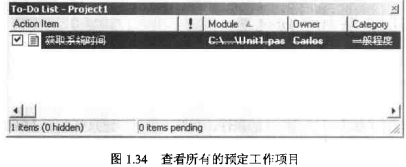


图 1.34　查看所有的预定工作项目

1.6　如何获取更多的相关信息

如何学好一种程序语言，尤其是面向对象的程序语言，您必须知道哪里可以得到帮助，以下将和大家分享个人学习的经验，告诉您哪里可以得到更多的相关信息。

1.6.1 Delphi 6 使用手册

如果您有原版的 Delphi 6 光盘，可以在光盘目录中找到一个 Documentation 的目录，其中有 3 个.pdf 的文件，这就是 Delphi 6 的电子书，包含 Quick Start、Developer's Guide 及 Object Pascal Language Guide 等三本电子书。如果您要查看这些电子书的内容，请先安装 Acrobat Reader 这套软件。安装也很容易，只要运行“Ib6\Adobe\rs40eng.exe”即可。如果您已经正确安装了 Acrobat Reader 这套软件，您就可以运行“Documentation\Qs.pdf”，其画面如图 1.35 所示。

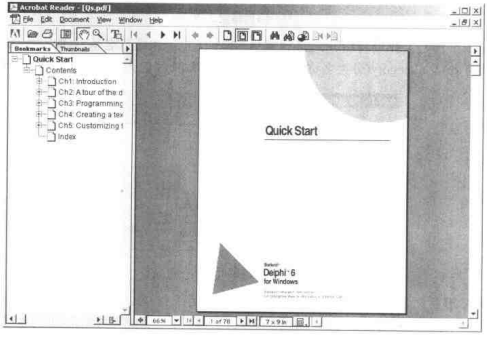


图 1.35 Delphi 6 电子书

1.6.2 Delphi 6 联机帮助说明

如果您不想看这些电子书，您也可以通过联机帮助说明来获取相关信息。您有三个方法去查阅这些联机帮助说明。

1. 使用【F1】功能键

 如果您对于哪个组件或组件的属性、事件不了解，您只要按下【F1】功能键即可获得帮助。以下用四种情况来说明【F1】功能键的使用方法。

 - 假如您不了解什么是窗体(Form)，您只要在 Form 的上方左键点选一下，并按下【F1】功能键即会出现联机帮助说明功能，如图 1.36 所示。

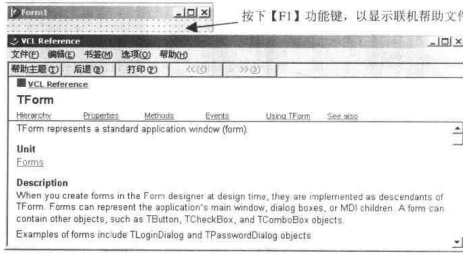


图 1.36　窗体【F1】功能键

- 假如您不了解 Form 的 Caption 属性是什么、要如何设置，您只要先点选对象检查器的 Caption 属性字段，再按下【F1】功能键即会出现联机帮助说明功能，如图 1.37 所示。



图 1.37　对象属性【F1】功能键

- 假如您不了解 Form 的 onClick 事件是什么、要如何设置，您只要先点选对象检查器的 onClick 事件字段，再按下【F1】功能键即会出现联机帮助说明功能，如图 1.38 所示。

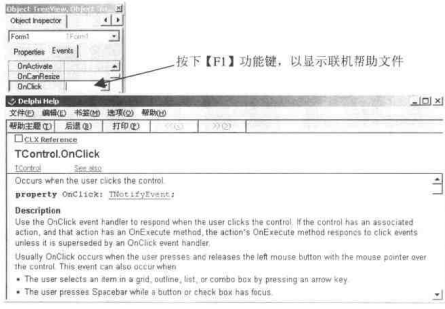


图 1.38　对象事件【F1】功能键

假如您在程序编辑过程中对某个函数不了解，您只要在函数名称所在位置按下【F1】功能键即会出现联机帮助说明功能，如图 1.39 所示。

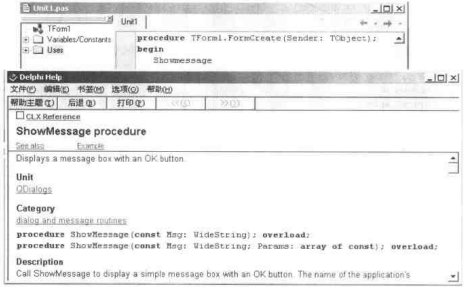


图 1.39　程序编辑的【F1】功能键

2. 使用下拉菜单(或快捷键)

如果您不使用【F1】功能键，您同样可以通过下拉菜单及加速栏的联机帮助说明按钮来调用联机帮助功能。如图 1.40 所示。



图 1.40　联机帮助说明

3. 直接运行 Help File

您除了可以使用上述的方式之外，还可以直接运行 Delphi 6 所提供的 Help File。Delphi 6 提供了数十个 Help File，您可以利用 Windows 98/Me/2000 所提供的文件搜索功能去查找扩展名为 hlp 的文件。

1.6.3　*Borland Web Site*

除了上述的联机帮助文件外，您也可以通过网络直接到 Borland 或是 INPRISE 的首页，获取 Delphi 6 的最新信息，网址如下：http://www.borland-inprise.com，其首页如图 1.41 所示。

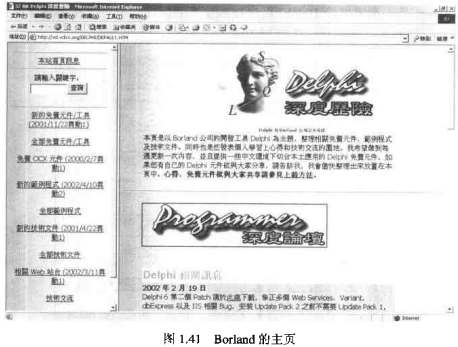


图 1.41　Borland 的主页

1.6.4 Delphi 深度历险网站

另外还有一个网站是学 Delphi 的人一定要去逛逛的，那就是深度历险网站，这是一个非常有深度的网站，也是 Delphi 初学者的殿堂，网址是 http://vcl.vclxx.org/DELPHIGB/DEFAULT.HTM，如图 1.42 所示。



图 1.42 深度历险网站

1.6.5 加入 Delphi Chart

Delphi Chart 是一个提供给 Delphi 用户技术交流的园地，您可以在那里学习到更多前辈的经验与技术。您可从“深度历险”网页中进入，如图 1.43 所示。



图 1.43 加入 Delphi Chart

第 2 章

Delphi 6 程序结构

本 章 目 标

本章主要说明 Delphi 6 的程序结构，我们从最简单的范例开始说起，以便让大家了解什么是工程(Project)，并且为大家说明如何利用工程管理器(Project Manager)来管理您的工程。最后再为大家说明如何有效地利用对象库(Component Repository)以缩短程序开发的时间。

本 章 主 题

2.1 三秒钟完成第一个程序

2.2 了解工程 (Project)

2.3 利用工程管理器(Project Manager)

2.4 工程选项设置 (Project Options)

2.5 利用对象库 (Object Repository)

2.1 三秒钟完成第一个程序

经过了上一章的介绍，相信您对 Delphi 6 已经有了初步的认识，也应该对 Delphi 6 整合性开发环境有相当的了解。本节将带领各位读者完成您第一个 Delphi 6 的程序，虽然是个很简单的程序，但希望通过这个简单的程序，让您了解 Delphi 6 的程序结构。

范例 2-1

目的：通过创建一个简单的程序，说明 Delphi 6 开发程序最基本的操作，并让您了解 Delphi 6 的程序结构。

① 启动 Delphi 6。当 Delphi 6 完全打开后，您将会看到图 2.1 所示的画面。此时 Delphi 6 已经帮您默认打开一个名称为 Project1 的工程，在此工程内也默认了一个名称为 Form1 的窗体。

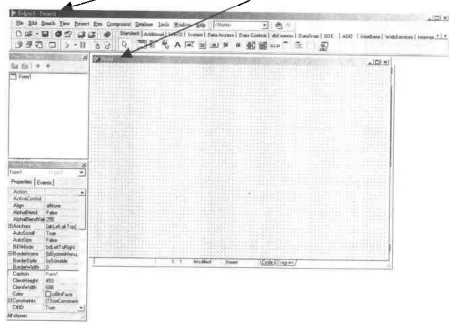


图 2.1 Delphi 6 完全启动后的画面

② 现在 Delphi 6 已经帮我们建立一个包含一个窗体的工程，而这就是构成窗口程序的最基本需求，所以在运行程序之前，我们先将程序另存新文件到磁盘中。首先请选择菜单栏中的“File/Save Project As…”，并输入单元文件(unit)及工程文件(project)的文件名称，这里单元文件以 exUnit1.pas 为例，而工程文件以 ex1.dpr 为例。工程文件的名称也就是将来产生可执行文件(.EXE)的名称。如图 2.2 所示。

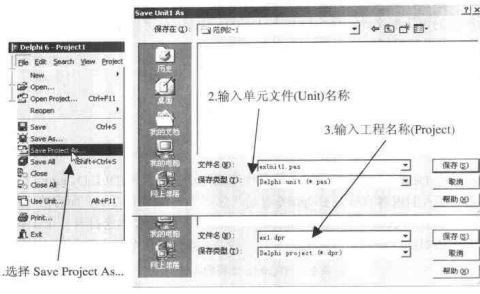


图 2.2　保存第一个工程到指定目录

3 确定工程已经正确保存后，按下【F9】功能键(或按下加速栏的运行程序按钮)来运行程序。这时您的第一个程序已经完成了。其中 代表窗体默认的图标、Form1代表窗体默认的 Title 属性、 代表最小化按钮、 代表最大化钮、 代表关闭按钮，如图 2.3 所示。

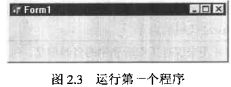


图 2.3　运行第一个程序

您可能会很怀疑地想要问，“这真的是一个程序吗?但是我什么都没做!”。我很慎重地回答您，这真的是一个程序，因为它具有窗口程序的基本要件：关闭、放大、缩小窗口等功能。所以如果您还怀疑的话，请您到存放工程的地方，用资源管理器查看一下吧!其中 exUnit.pas 是单元文件，ex1.dpr 是工程文件而 ex1.exe 则是工程所产生的可执行文件。如图 2.4 所示。

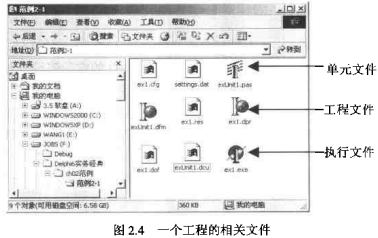


图 2.4　一个工程的相关文件

没错吧？您会看到一个文件为 Ex1.exe 的可执行文件，那就是将整个工程经过编译及链接之后所产生的可执行文件，您在图标上 Double Click 就可以直接运行了，不再需要经过 Delphi 6 的集成性开发环境。

2.2 了解工程(Project)

2.2.1 什么是工程(Project)

“工程”在 Delphi 6 里扮演着组合应用程序或是动态链接库(DLL,Dynamic Link Library)集合的角色。工程的内容有些是 Delphi 6 自动帮您产生的(如范例 2-1)，而有些可能是您自己加入的程序、过程(Procedure)或函数(Function)。以范例 2-1 为例，当您打开一个工程时，Delphi 6 会自动产生相对的代码和一些必要的文件。表 2.1 就针对这个基本工程的内容加以说明。

表 2.1 Delphi 6 工程的各种文件

扩展名	说明
*.DPR	Delphi 6 的工程文件(Project File)，这个文件包含了 Delphi 6 的程序运行顺序，并且指明窗体文件或是程序文件的实际路径位置
*.DFM	Delphi 6 将每一个窗体用一个文件来保存其内容与设置，这个文件说明了该窗体使用了哪些组件及组件的属性等信息。因此这种文件可能会不止一个
*.PAS	每一个窗体会对应一个.PAS 文件，但有.PAS 文件不一定会有对应的窗体，这种文件通常是用来记录 Object Pascal 代码的内容
*.RES	这个文件是 Delphi 6 用来保存工程所使用的资源信息，包含工程所用到的图标(Icon)、图形(Bitmap)等，文件名通常与工程名称一样
*.DOF	这个文件存储这个工程所用到的各种编译器的设置，文件名通常与工程名称一样
*.CFG	这个文件用来保存有关工程 configuration 的信息，它是一个 compiler-readable config file，文件名通常与工程名称一样
*.~??	当各类文件有修改发生，系统会自动加以备份，它就是备份的文件

2.2.2 工程文件(.DPR)

Delphi 6 会为每一个工程产生一个扩展名为.DPR(Delphi Project)的文件，这个文件包含了 Delphi 6 的程序运行顺序，并且指明窗体文件或是程序文件的实际路径位置。一般来说，这个文件都必须由 Object Pascal 程序所组成，当您编译或运行工程，Delphi 6 的编译器会自动产生与 Delphi 6 同文件名的可执行文件或动态链接库，其扩展名分别为.EXE 或.DLL。一般的情况下您应该不会去更改这个文件的内容，但如果您想制作一个类似于 Delphi 6 打开时所产生的 LOGO 画面，或是您想控制程序的试用期间，那就可能会去更改它的内容。至于如何更改，我们会在后面的章节为大家介绍。现在我们来查看一个工程文件究竟包含了哪些代码，请依照下列步骤来操作：

1 创建新的工程(可使用菜单栏中的“File/New Application”完成)。

② 选择菜单“Project/View Source”。如图 2.5 所示。



图 2.5 查看工程文件代码的内容

③ 此时，代码浏览器(Code Explorer)中会出现一个“Project1”的页标签，并显示工程文件的代码内容，如图 2.6 所示。

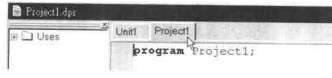


图 2.6 查看工程文件代码内容

这里以 Delphi 6 默认的空白工程为例，其内容如下：

Delphi 6 默认的空白工程

```
Program Project1;
Uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

现在我们来看看上面的代码究竟代表什么意思：

- Program

 program 是声明工程的标识符，由 program 所声明的工程是一个应用程序，如果是由 library 所声明的工程则是一个动态链接库。而在 program 保留字之后的 project1 则是这个工程的名称，这是由 Delphi 6 所默认的名称，您当然可以修改它，但您不能直接在代码浏览器(Code Explorer)修改它。如果我在代码浏览器中直接将 Project1 改为 MyProject1，并按【F9】功能键运行工程，此时系统会出现 File not found:'MyProject1.RES'的错误信息，如图 2.7 所示。

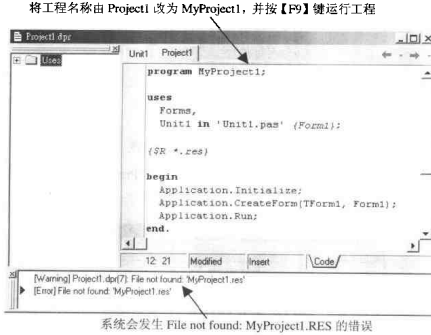


图 2.7　不当的更名方法会产生错误

如果您想更改工程名称，那么请按照下面的步骤来操作即可：

1. 选择菜单栏中的“File / Save Project As…”。
2. 此时会出现一个“Save Project1 As”的对话框，在“文件名”输入新的工程名称，并单击 保存(S) 按钮即可。
3. Delphi 6 会自动将工程名称更改为新的文件名称。如图 2.8 所示。



图 2.8　正确的重命名方法

Uses

uses 子句主要是用来告诉 Delphi 6 的程序编译器有哪些文件被链接到文件中，其中包含非窗体(not Form)的程序单元及窗体(Form)所对应的程序单元(Unit)，请参见图 2.7。在 uses 中 Forms 是一个标准程序单元的标识符，只要您的程序用到 TForm 对象，系统便会自动加进来。保留字 in 是用来告诉 Delphi 6 的程序编译器应该到哪里找到原始的程序文件。如果没有路径，就表示在同一个目录。而{Form1}表示该单元是一个具有窗体的单元文件。如果一个单元文件没有对应的{窗体名称}，就表示它是一个不含窗体的程序单元，它可能只是一个简单的 Object Pascal 程序文件。如果您想建立一个不含窗体的程序单元文件，您只要进行下列步骤即可：

1. 建立新的工程(可使用“File/New Application”来完成)。
2. 选择菜单栏中“File / New…”。
3. 在“New Items”窗口的“New”页标签中选择“Unit”的图标，按下即可，如图 2.9 所示。

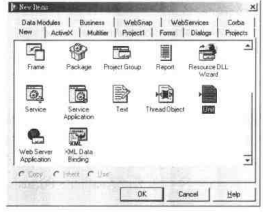


图 2.9 添加一个单元文件

4. 此时在代码浏览器中找到一个新的单元文件，其默认名称为 Unit2。

{$R *.RES}

$R 是与编译器有关的指令，而{$R *.RES}是用来说明与工程有关的资源文件。若是用*.RES，就表示这个资源文件与工程有相同的名称。

begin 与 end

由 begin 及 end 所构成的块是工程主要的程序块。其中“Application.Initialize;”代表工程的初始化，它负责完成一个应用程序应有的资源规划；“Application.CreateForm(TForm1, Form1);”负责转入所对应参数的窗体，其中 TFom1 代表对象类别，Form1 代表窗体对象的名称，这样的语句可能不只一个，它会根据应用程序包含的 TForm 对象而决定，但并不是每一个应用程序的 TForm 对象都会在这里出现，只有运行时由系统自动产生(AutoCreate Form)的 TForm 对象才会出现在这里。反之，如果是由用户自行控制(Available-Form)的 TForm 对象是不会在这里出现的。“Application.Run;”语句代表工程的运行点，也就是说前面的语句

都只是将数据加载内存(RAM)而已，这里才是真正的运行点。以下将说明如何设置TForm对象是自动产生，还是由用户自行控制产生。

1. 创建新的工程。
2. 如果您利用“Project/View Source”来查看工程文件的内容，那么您会发现整个工程只调用一次CreateForm操作方法，因此系统只会自动创建一个窗体Form1。
3. 如果您想再创建一个新的窗体，您可以利用“File/New Form”来产生一个新的窗体。
4. 在您添加窗体之后，请再一次利用代码浏览器查看工程的内容，此时Uses子句与创建窗体对象的语句，会同时加入第二个窗体的信息，其内容如下所示。

```
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

5. 步骤4所查看的内容代表系统在启动时会将两个窗体加载内存(RAM)。如果您不想这么做，可以利用菜单栏中“Project / Option...”弹出“Project Options”窗口，在“Forms”页标签中将Form2对象设为“Available Forms”。在这里顺便说明一下，我们不可将所有的TForm对象都设为“Available Forms”，否则此工程是无法被运行的，如图2.10所示。

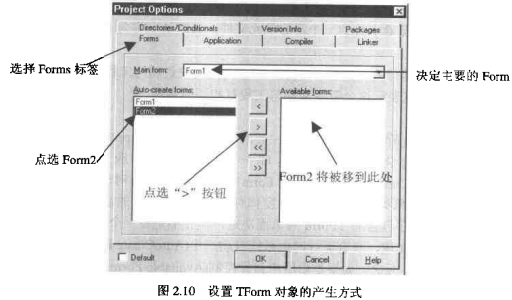


图2.10 设置TForm对象的产生方式

6 再利用代码浏览器查看工程内容，此时创建窗体对象的语句只有“Application.CreateForm(TForm1, Form1);”。这代表系统会自动产生 Form1，而 Form2 则必须靠您自己动态产生，至于如何动态产生 Form2，在以后的章节会有详细的说明。

2.2.3　窗体文件(.DFM)

前面曾经提过一个工程可能具有多个 TForm 对象。虽然在 Delphi 6 中您是利用可视化工具来设计程序，换言之，就是在窗体上加入新的对象，但事实上 Delphi 6 会将这些信息以二进制的方式写到窗体文件中，窗体文件通常与单元文件具有相同的文件名，但其扩展名为.DFM(Delphi Form)。由于信息是用二进制的方式保存，所以您无法利用一般的文本编辑程序来读取，但是话又说回来，这个文件您不需要去更改它的内容。如果您坚持想要看看它的内容，那您就跟着我做吧！

1 在 TForm 对象上单击鼠标右键，在菜单中选择“View As Text”，如图 2.11 所示。

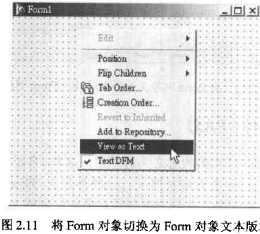


图 2.11　将 Form 对象切换为 Form 对象文本版本

2 在代码浏览器中会出现一个文件名为“Unit1.DFM”的文件，其内容如图 2.12 所示。

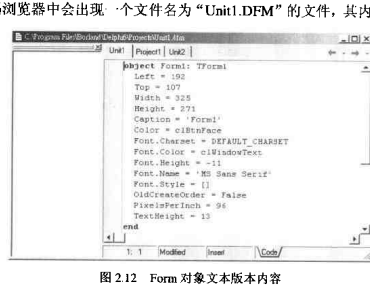


图 2.12　Form 对象文本版本内容

3 如果您想切换为窗体对象的话，请在代码浏览器中“Unit1.DFM”文件内容上利用鼠标右键菜单，选择“View As Form”，进行切换的工作，如图 2.13 所示。

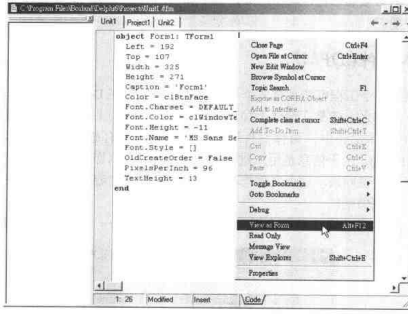


图 2.13 将 Form 对象文本版本切换为 Form 对象

2.2.4 程序单元文件(.PAS)

由 unit 保留字开始到 end 结束的块称之为程序单元文件，它是一般程序内容的主要文件。前面有提过，程序单元文件可能会有两种情况，一种是包含 TForm 对象，另一种是不包含 TForm 对象。在前面的“工程文件”部分中也曾经示范如何创建一个不包含 TForm 对象的单元文件，因此这里只针对包含 TForm 对象的单元文件加以说明。当您创建一个新工程，Delphi 会自动帮您产生一个单元文件，其内容如下：

Delphi 6 默认的单元文件内容

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
Type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

```
implementation
{$R *.DFM}
end.
```

现在我们来看看上面的代码究竟代表什么意思：

1. unit 标识符

unit 是程序单元的标识符，而 Unit1 则为这个单元文件名。

2. Interface 标识符

interface 是在单元中的一个 Section，这个 Section 主要是用来声明常数(constants)、变量(variables)、过程 (procedures)、函数(functions)或是定义一些类型(types)。

3. uses 标识符

uses 语句主要是将此单元所需要的链接库包含进来，例如我们在 Form 上加入一个 Button 的对象，只要编译(Compile)或运行(Run)此工程，那么 Delphi 6 会自动加入 StdCtrls 链接库，因为 Button 对象是被定义在 StdCtrls 链接库的。但是如果您删除某个组件，Delphi 6 并不会自动删除其所引用的链接库。如图 2.14 所示。

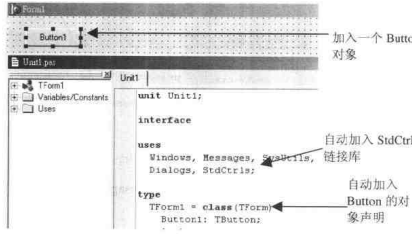


图 2.14　设计时期自动加入代码

4. type…end 区域

在 Type 区域中通常是用来声明这个 Unit 所用到的类型，以范例 2-1 来说，因为 ex1Unit 只用到一个 Form 对象，所以您只会看到 TForm1 的声明。在 Type 中包含了 Private 及 Public 两个小区域。简单地说，放在 Private 区域中的变量、过程或函数，其他的 Unit 是无法存取的。相对的，放在 Public 区域的变量、过程或函数，其他的 Unit 是可以存取的。而“end;”则表示是 Type 区的结束符号。在这个例子中，目前在 TForm1 中只包含了一个 Button1 的对象(参见图 2.14 所示)。

5. var 标识符

在 var 区域中同样可以用来声明变量、过程或函数，但以这个例子而言，它只用来作为 Form 对象的声明而已。

6. implementation 区域

implementation 是单元中的另一个 Section，这个 Section 主要用来定义(define)

在 interface section 所声明的过程(procedure)和函数(function)。换言之，这里才是真正撰写 Procedure 及 Function 代码的地方。用一个例子来说明 Implementation Section。

范例 2-2

目的：试作简单的程序，让您了解我们真正编写代码的位置。

1. 创建新的工程(可使用“File / New Application”来完成)。
2. 在 Form 上加入一个 Button 对象。
3. 在 Form 的 Create 事件处理过程中加入“ShowMessage('窗体被创建了');”等代码。，如图 2.15 所示。

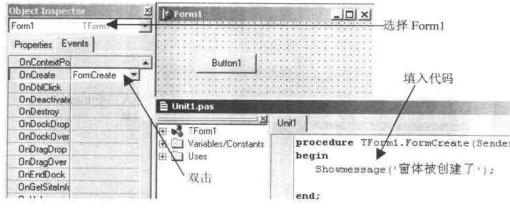


图 2.15 加入 Form 的 Create 事件内容

4. 在 Button 的 Click 事件处理过程中加入“ShowMessage('按钮被按下了');”等代码，如图 2.16 所示。

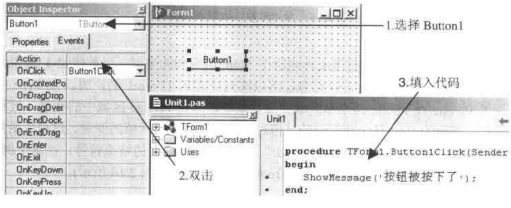


图 2.16 加入 Button 的 Click 事件内容

5. 保存工程。
6. 按【F9】功能键，运行程序。

我想这个程序的运行结果应该是很容易理解的，当 Form1 对象创建时会引发 Form1 对象的 onCreate 事件，因此系统会出现一个“窗体被创建了”的对话框，您必须单击“OK”按钮。在 Form1 对象被创建之后，当您按下 Button1 对象，系统会触发 Button1 对象的 onClick 事件，并出现一个“按钮被按下了”的对话框。这样的程序就是所谓的事件驱动。以后的程序我们会循序渐进地介绍相关的程序编写技巧。我们做这个例子主要目的是要说明 Implementation Section，现在我们就来看看 Unit1 的所有内容吧。

范例 2-2 部分代码

```
Implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
   ShowMessage('窗体被创建了');
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('按钮被按下了');
end;
end.
```



在 Implementation Section 中的{$R *.DFM}您可千万别将它删掉，因为 Delphi 6 可是利用它来连接 TForm 对象的二进制窗体文件，如果不见了，可是会出错的！

2.3　利用工程管理器(Project Manager)

当您工程中有多个 TForm 对象或单元文件时，您就应该利用工程管理器(Project Manager)来帮您管理。以下就用一个范例来说明如何使用工程管理器。

2-3

目的：创建新的工程，并学习如何使用工程管理器来帮助我们管理窗体及单元文件。

1. 创建新的工程(可使用“File/New/ Application”来完成)。
2. 利用菜单栏中的“File / New /Form”创建第二个 TForm 对象。
3. 选取菜单栏中的“View / Project Manager”，打开工程管理器，如图 2.17 所示，在工程管理器中可以利用加入新的或已经存在其他程序的单元文件，当然您也可以利用来将已经存在这个工程的单元文件加以删除。最重要的是您可以利用单元文件图标，或窗体文件图标快速弹出单元文件或是 TForm 对象。在图 2.17 中，您还可以发现 Unit2 与 Unit1 的路径名称不一样，那是因为 Unit2 尚未保存，所以会出现系统默认的路径。

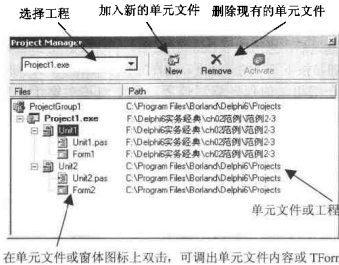


图 2.17　工程管理器(Project Manager)

④ 接着示范如何加入一个新的单元项目。请按下，打开一个名为“Items”的窗口，

⑤ 选择“DLL”的图标，再按下“确定”按钮。此时 Delphi 6 便自动创建一个名称为 Project2 的动态链接库(DLL)的工程，如图 2.18 所示。

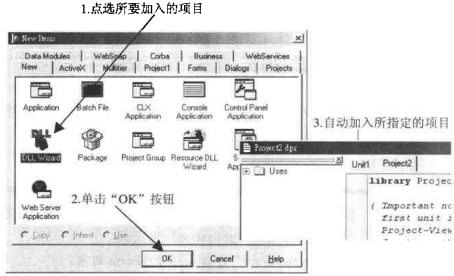


图 2.18　加入新的工程文件

⑥ 如果您想将刚刚加入的 DLL 文件删除，您只要在工程管理器(Project Manager)中用鼠标左键点选刚刚所加入的 DLL 文件图标，再按下即可。此时会出现一个“Confirm”的对话框，如果您确定要删除，只要单击“Yes”按钮即可，如图 2.19 所示。

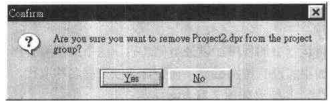


图 2.19　删除单元文件确认对话框

2.4　工程选项设置(*Project Options*)

在 2.2 节中曾经提过工程选项设置文件(.DOF). Delphi 6 的工程选项设置文件是一个文字文件，主要包含工程选项的最新设置值，文件名通常与工程名称一样，其扩展名为(.DOF, Delphi Options File)。您可以利用工程选项对话框(Project Options)来更改工程的各项设置。

当您打开一个新工程或现有工程后，只要您选择菜单栏中的“Project / Options”，此时 Delphi 6 会自动打开 Project Options。只要您在 Project Options 对话框中更改任何的设置，系统会在目前工程的目录下为您打开一个与工程同文件名，但扩展名为.DOF 的工程选项文件，这些设置必须重新启动工程才会生效。这些设置可分为七大类，如图 2.20 所示，分述如下：

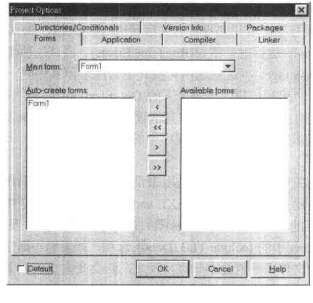


图 2.20　工程选项设置

1. 窗体(Forms)页标签
 这个画面您可能似曾相识。是的，没错，我们曾在第 2.2.2 节中详细地介绍过，它可以让您更改工程的主要窗体及设置工程运行时的创建顺序，或是设置自动创建或手动创建的窗体，如果您忘记了，麻烦您再翻回第 2.2.2 节，好好复习一下。
2. 应用程序选项(Application)页标签
 在应用程序选项(Application)中主要是设置工程的标题(Title)、帮助文件(Help)及图标(Icon)，可以让您将这些设置设为默认(Default)，换句话说，只要您设置一次，就

可以分享给以后新创建的工程使用。

- Title 选项

 如果您将运行中的程序加以最小化之后，这个文字将会出现在任务栏上的小图标中，如果您没有特别指定，它会自动以工程名称取代。我们将它改写为“测试 Title”，其结果如图 2.21 所示。



更改前　　更改后

图 2.21　应用程序 Title 选项设置

- Help 选项

 这个选项是为了连接帮助文件(.HLP)之用。您可以直接输入文件名，也可以利用 [redacted] 来选取帮助文件。

- Icon

 在 Title 的范例中所显示的[redacted]是 Delphi 6 所默认的图标，您可以利用 [redacted] 加载指定的图标文件(Icon)。Delphi 6 有一些图标文件可供您使用，通常它们被放在“C:\Program Files\Common Files\Borland Shared\Images\Icons”的目录之下。

3. Compiler 页标签

 在 Compiler 页标签主要让您设置有关运行代码的产生(Code Generation)、语法选项(Syntax Options)、运行时期错误(RunTime Errors)、调试选项(Debugging)及编译时期的信息选项(Messages)。这些选项其实并不常用，但其中有一二项您可以稍微了解一下。例如：“运行代码的产生”下的 Optimization 选项，它可以将程序进行最佳化的处理。还有像“编译时期的信息选项”下的 Show Hints 及 Show Warnings 可以在编译时期显示提示信息及显示程序中的警告信息。

4. Linker 页标签

 链接时期的相关选项页标签主要是用来设置链接程序文件的相关选项，如 Map File、可执行文件及动态链接库的选项(EXE and DLL Options)、链接器的输出选项(Linker Output)及内存的大小(Memory Sizes)等。其中在可执行文件及动态链接库选项(EXE and DLL Options)中的 Include TD32Debug 选项，如果您不选它，您的可执行文件及动态链接库的文件大小就会小一点，反之，则会变大。但天下没有白吃的午餐，如果您没有选它，您就会少了调试(Debug)的相关信息。

5. Directories/Conditionals 页标签

 在目录(Directories)与条件(Conditionals)选项页标签中，主要是提供您掌握工程编译结果与输出有关的选项，包含指定编译、链接和产生应用程序所需要的资源所在目录位置或路径。其中有三个可能会是您比较常用到的，说明如下：

 - 输出目录(Output Directory)

 输出目录是指工程经过编译之后所产生的可执行文件或动态链接库所存放的目录。如果您是利用文件服务器(File Server)的架构来运行 Delphi 6 的话，那么这个选项您就必须设置 Local 的机器，因为通常您对服务器都只有只读的权限

而已。

- 单元文件输出目录(Unit Output Directory)
 单元文件输出目录与输出目录(Output Directory)有相同的意义，唯一的不同是此目录针对的是程序单元(Unit)。
- 搜索路径(Search Path)
 搜索路径是用来指定编译程序所需要的函数库路径，通常它是指存放非 Delphi 6 的函数库路径。

6. Version Info 页标签
 版本信息(Version Information)页标签是让您指定此工程与版本有关的选项，如果您想将这些信息加入此工程，只要您点选“Include version information in project”即可。
7. Packages 页标签
 Packages 页标签的选项可以让您设置与此工程有关的 Package 信息。什么是 Packages 呢？在 Delphi 6 本身已经运用到 Package。它是一种特别的动态链接库，在运行时期它允许应用程序间共享代码，可以大大节省系统资源与硬盘内存的空间，您同样也可以创建属于自己的 Package，使用 Package 要比传统的程序开发方法来得有效率。Delphi 6 中提供在 IDE 中的组件都是通过 Package 而来，您可通过菜单选择“Component / Install Packages...”调用“Project Options”对话框得知相关信息，例如组件面板页标签中的标准组件(Standard)都属于 Dclstd40.bpl，与数据库有关的组件(DataBase Component)则属于 Dcldb40.bpl。

2.5 利用对象库(Object Repository)

2.5.1 使用对象库的对象

Delphi 6 除了提供组件面板上的组件之外，另外提供对象库(Object Repository)供您重复使用自定义的组件。对象库除了个人使用外，更可以运用到开发团队上，通过因特网(Internet)进行资源共享。您可以利用三种方式来使用：复制(Copy)、继承(Inherit)及使用(Use)。

1. 复制(Copy)
 您可以复制现成的窗体或工程来使用，由于是复制因此您所复制之后的副本与对象库的原始对象无关。
2. 继承(Inherit)
 指套用现成的窗体或工程，但当您重新编译应用程序时，如果您有修改，它会一并影响到应用程序所继承的窗体或工程上。
3. 使用(Use)
 是指您可以使用现在的窗体或工程，只要您改变这些使用项目，这些改变也会同时发生在其他的应用程序中。

范例 2-4

目的：本范例的目的是通过创建一个密码输入对话框，来示范如何使用对象库的窗体节省程序开发的时间。

步骤 1 创建的新工程。

步骤 2 利用菜单栏中的“File / New...”弹出“New Items”窗口。选择“Dialogs”页标签中的“Password Dialog”图标，并设置为“复制模式”，再单击“OK”按钮，如图 2.22 所示。

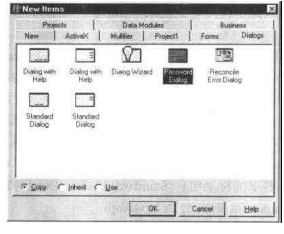


图 2.22 利用对象库

步骤 3 在 TForm 对象(打开工程所默认的 TForm)上加入一个 TButton，Caption 设为“打开密码输入框”，并在 TButton 对象的 onClick 事件内容中加入“PasswordDlg.ShowModal;”代码。因为此密码对话框的默认名称为 PasswordDlg，其继承自 Tpassword 对象。

步骤 4 别忘了在 Uses 子句中加入新的密码单元文件(默认为 Unit2)。有下列两种方法来加入，第一个方法是将 Units 加到 Uses 标识符后，另一个方法是将 Uses Units 加入 implementation 中。

```
                          unit Unit1;

implementation            interface

uses Unit2;               uses
                            Windows, Messages, SysUtil
                            StdCtrls,Unit2;

        方法一                      方法二
```

注意 但实际上，我们也可以用 Delphi 6 IDE 所提供的功能来完成这个项目，以下将以 Unit1 引用 Unit2 为例。首先，请用菜单栏中的“View/Units”将程序编辑器切换至 Unit1。再利用菜单栏中的“File/Use Units”将 Unit2 供 Unit1 引用，如图 2.23 所示。

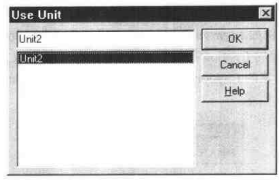


图 2.23　Use Unit 对话框

⑤ 保存工程。

⑥ 按【F9】功能键运行工程，此时若按下“打开密码输入框”按钮，则会自动打开密码输入对话框，如图 2.24 所示。



图 2.24　利用对象库范例运行结果

2.5.2　创建自己的对象库

如果您有一些程序需要一再地出现在各个工程，您可以使用复制的方式重复使用该单元文件，但是您有更好的选择，那就是将自己所写的程序加到对象库(Object Repository)中。例如上面所使用的“密码确认框”，由于 Delphi 6 在对象库中所提供的是英文版，但您的用户要求中文化，因此您只要作少许的修改，再将它存回对象库中，那么您就可以重复使用这个“中文化的密码确认对话框”了。

范例 2-5

目的：本范例的目的是通过修改对象库中现存的对象，以创建属于自己的对象库。

① 关闭所有文件(菜单栏中的“File / Close All”)。

② 选择菜单栏中的“File / New/Other”，打开“New Items”窗口，从“Dialogs”页标签中，选择“Password Dialog”图标，并单击“OK”按钮(可参考范例 2-4)。

3 此时出现一个英文版的对话框，您只要将 Form 对象上的所有对象的显示文字，更改为中文即可。例如 Form 对象的 Caption 属性值“Password Dialog”改为“密码确认”，Label 对象的 Caption 属性值“Enter password:”改为“请输入您的密码”，且两个 Button 对象的 Caption 属性也分别从“OK”及“Cancel”改为“确定”与“取消”，如图 2.25 所示。

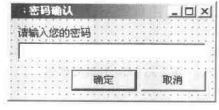


图 2.25 中文化的界面

4 保存文件。

5 在“密码确认”框中按鼠标右键，选择 Add To Repository…。

6 此时出现对象库新增对象的编辑器(Add to Repository)。并填入适当的信息，如图 2.26 所示。

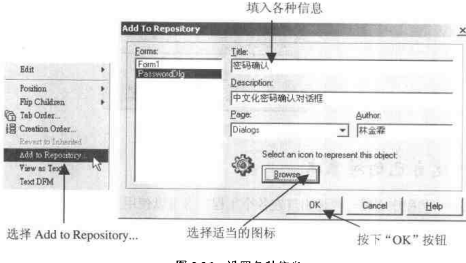


图 2.26 设置各种信息

7 此时您可以检查一下对象库，看看这个中文化的密码确认对话框是否出现在“Dialogs”页标签中，如图 2.27 所示。

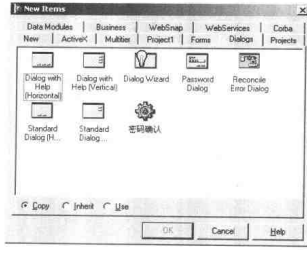


图 2.27　属于自己的对象



如果您想删除对象库的内容，那么您可以选择菜单栏中的“Tools/Repository”，此时系统会打开“Object Repository”对话框，您可以从此对话框中删除指定的对象。以下示范如何删除先前所加入的“密码确认”对象。

1. 指定 Pages，请选择 Dialogs Page。
2. 指定 Objects，请选择“密码确认”对象。
3. 单击“Delete Object”按钮，并按下确认对话框中的“Yes”按钮。
4. 单击“OK”按钮，结束 Object Repository 对话框。

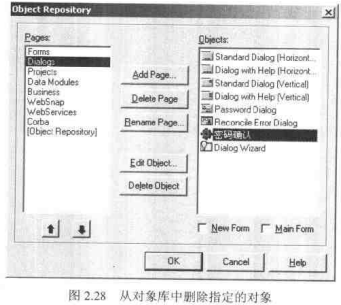


图 2.28　从对象库中删除指定的对象

第 3 章

Delphi 6 对象共同属性

本章目标

面向对象程序设计最重要就是利用各个对象的属性，然而很多对象可能会有一些相同的属性。本章的目标便是为大家说明这些对象的共同属性。我们将对象共同属性分为与位置或大小有关、与显示或外表有关、与 Parent 有关、与引导有关及与拖动有关等属性。在本章最后将说明面向对象程序设计几个重要的名词，如 Parent、Owner、Sender 与 Self 等。

本章主题

3.1 与位置或大小有关的属性

3.2 与显示或外表有关的属性

3.3 与 Parent 有关的属性

3.4 与引导有关的属性

3.5 与拖动有关的属性

3.6 设置共同属性

3.7 Parent、Owner、Sender 与 Self

记得前面曾经提过，Form 也是一个 Delphi 6 的对象。但它可不是 Delphi 6 所定义的最初对象喔，它是由最初的 TObject 一直一路继承下来的，如图 3.1 所示。它是继承了 TObject、TComponent 及 TControl 的基本特性而来，Delphi 6 很多组件也都是这样来的。在图 3.1 只是简略地说明其继承的关系，事实上您可以通过 On Line Help 查到更详细的关系。

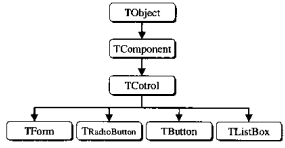


图 3.1　简单的对象继承关系

既然 TForm 也算一个对象，那么我们就来谈谈大部分的对象有哪些共同的属性。大部分对象的共同属性可以粗略地分为“与位置或大小有关的属性”、“与显示或外表有关的属性”、“与 Parent 有关的属性”、“与引导使用有关的属性”及“与拖动相关的属性”等 5 个部分来说明。

在没有正式说明这五大类的共同属性之前，我们必须先来谈对象一个很重要的标识属性 (Name)，这个属性是无法共享的。在人类社会几乎每个人都会有一个名字，通常这个名字在某些场合可以被当作是惟一标识的信息，但是在某些场合却可能不是这样的。例如“樱木花道”在他家可能是惟一的，但在学校却可能同时出现很多个“樱木花道”。在 Delphi 6 的对象社会里也同样允许这样的情况发生。如图 3.2 所示，在应用程序中的每个 TFrom 对象都必须具有惟一的 Name 属性，但在个别的 TForm 中却允许分别有着相同 Name 属性的 TButton 对象。

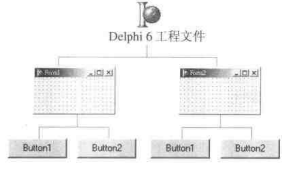


图 3.2　Delphi 6　Name 属性关系图

另外，若您不小心在同一个 Form 中输入相同 Name 属性为 Button1 的 TButton 对象，此时 Delphi 6 会显示错误信息，并将该 Name 属性恢复成原来的。如图 3.3 所示。

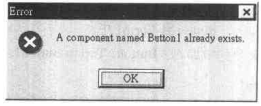


图 3.3　输入相同 Name 属性显示的错误信息

虽然您无法在同一个 TForm 对象下创建两个名称完全相同的对象，但您却可以在 TForm 对象下创建一个与 TForm 对象名称完全相同的对象。以图 3.2 为例，您可以将 Form1 对象下的 Button1 对象名称改为 Form1。虽然 Delphi 6 允许您这么做，但是笔者还是要建议您尽量避免使用这种命名方法以免您自己混淆。

3.1　与位置或大小有关的属性

会影响对象位置或大小有关的属性共有四个，分别为 Height、Width、Top、Left 等，如图 3.4 所示。

- Height 属性是指一个对象的高度。
- Width 属性是指一个对象的宽度。
- Top 属性是指垂直(上下)的位置。
- Left 属性是指水平(左右)的位置。

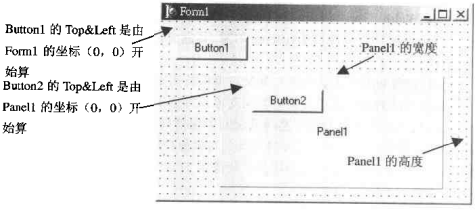


图 3.4　对象的位置与大小

范例 3-1

目的： 如果您想在设计时期修改对象属性，您只要在对象检查器的属性页标签中加以修改即可，在此不再赘述。本范例主要说明如何在运行时期修改对象的属性，以 Top 为例。

1 创建一个新的工程。如果您从前面开始一直都很用心的阅读，那一定会知道，刚启动 Delphi 6 之后的画面，系统已经帮您启动了一个工程，并且内含一个窗体。但如

果您想重新创建一个新的工程，就得从菜单栏中的“File/New Application”下手了。

Step 2 从组件面板中加入一个 Button 到 Form1 中。

Step 3 在 Form 的 onClick 事件中加入“Button1.Top := Button1. Top+10;”的代码。

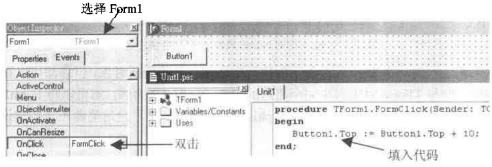


图 3.5　运行时期修改对象的位置

注意

这段代码会做什么事呢？很简单，当程序在运行时，您用鼠标左键在 Form 上按一下则 Button1 便会往下走 10 点(pixel)。其中 Button1 是代表对象，而 Top 是代表属性，在面向对象程序的世界里，您要让某个对象做事，只要先指明“对象”并紧接“.”，再接“属性”或“事件”即可，如果您有语法看不懂的话，请参考附录 A 的 Object Pascal 语法。

Step 4 保存工程(方法请参考范例 2-1)。

Step 5 按下【F9】功能键，运行程序，并且在 Form 上按下鼠标左键看看吧!!

除了 Height、Width、Top 与 Left 等四个属性外，Delphi 另外提供了 Anchors 属性以供您作画面的排列。Anchors 属性继承自 TAnchors 对象，其包含 akTop、akLeft、akRight 及 akBottom 等值，分述如下：

- akTop 代表此对象的上边界将与其父对象保持固定距离。
- akLeft 代表此对象的左边界将与其父对象保持固定距离。
- akRight 代表此对象的右边界将与其父对象保持固定距离。
- akBottom 代表此对象的下边界将与其父对象保持固定距离。

为了更进一步说明 Anchors 属性的用法，笔者在 TForm 对象上摆放一个 TMemo 对象与 TButton 对象，如图 3.6 所示。



图 3.6　Anchors 属性使用说明一

TMemo 与 TButton 等对象的 Anchors 属性值都默认为[akLeft, akTop]，因此程序运行时，如果您改变 TForm 对象的大小，那么其结果如图 3.7 所示。

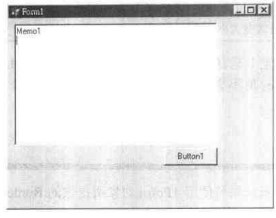


图 3.7　Anchors 属性使用说明二

这种情况是很难避免的，因为我们无法控制用户所使用的屏幕分辨率与操作行为，但是我们可以通过改变 Anchors 属性值来解决这个问题。以这个例子而言，我们可以将 TMemo 对象的 Anchors 属性设为[akLeft,akTop, akRight,akBottom]，也就是与父对象保持固定的四边边界距离；而 TButton 对象的 Anchors 属性设为[akRight,akBottom]，也就是与父对象保持右边界与下边界的固定距离即可。

3.2　与显示或外表有关的属性

一般的控制组件与外表有关的属性大致上可分为 BorderStyle、Color 及 Font 三种。同样的，这三种属性均可由设计时期或是由运行时期来修改。

- BorderStyle 属性是用来设置控制组件是否具有边框。
- Color 属性是用来设置控制组件的前景颜色。
- Font 属性是用来设置控制组件文字相关设置。

3.2.1　*BorderStyle* 属性

BorderStyle 属性是用来设置控制组件是否具有边框，以 TForm 对象而言，在 BorderStyle 的属性中共分为 bsDialog、bsSingle、bsNone、bsSizeable、bsToolWindow 及 bsSizeToolWin 等六种，见表 3.1。

表 3.1　BorderStyle 的种类

种类	意义
bsDialog	就像标准的对话框一样，不能改变大小只能关闭，而且也没有小图标
bsSingle	不能改变大小，只有细小的边框
bsNone	不能改变大小，且没有边框
bsSizeable	一般的窗口，可改变大小且有边框

续上表

种类	意义
bsToolWindow	与 bsSingle 相似，但标题栏的范围较小
bsSizeToolWin	可改变大小，但标题栏的范围较小

事实上这些属性在设置上是有一些限制的，例如您将一个 MDI Child 的 BorderStyle 修改为 bsDialog 或是 bsNone 是不会有效果的。以下将用范例 3-2 来说明如何将 Form 的 BorderStyle 改为 bsDialog。

范例 3-2

目的：本范例的目的主要示范如何使用 TForm 对象所提供的 BorderStyle 属性。

步骤 1 创建新工程。

步骤 2 通过对象检查器将 Form 的 BorderStyle 改为 bsDialog。

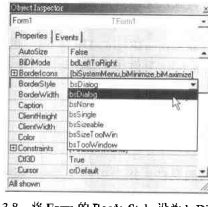


图 3.8 将 Form 的 BorderStyle 设为 bsDialog

步骤 3 保存工程。

步骤 4 运行程序，此时 Form 的右上角就只剩下一个结束按钮而已，其他类型的 BorderStyle 如图 3.9 所示。



图 3.9 各种 BorderStyle 的运行结果



当您将 Form 的 BorderStyle 设为 bsNone，运行时，如果您不知道怎么结束，请使用菜单栏中的“Run / Program Reset”来结束它，如图 3.10 所示，或按【Ctrl+F2】。可千万别重新启动计算机喔！



图 3.10　可利用 Program Reset 来结束程序

3.2.2　*Color* 属性

接着我们来谈谈 Color 这个属性。Color 这个属性主要是用来表示控制组件的前景颜色，同样可以通过设计时期及运行时期来改变其颜色。在设计时期要修改 Color 值，同样是利用对象检查器，详细请参见图 3.11。若想要在运行时期改变对象的 Color，可参考下面的例子。



图 3.11　设计时期改变对象颜色



3-3

目的：本范例的目的主要通过修改 TForm 对象的背景颜色为黄色来说明有关 Color 属性的设置方法。



创建新的工程。

Step 2 在 Form.onCreate 的事件中，加入“Color := clYellow; ”即可，如图 3.12 所示 。

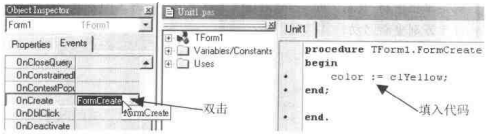


图 3.12 运行时期改变对象颜色

Step 3 保存工程。

Step 4 按【F9】功能键运行程序，并确认 Form 的颜色是否一开始就被改为黄色。

看完了上面的例子，您可能会有两个疑问：

1. 为什么要将代码写在 onCreate 的事件？

【解】：因为 onCreate 是创建 Form 时最早被运行的事件。

2. 为什么 Color 属性的前面没有写对象名称呢？

【解】：因为这些代码已经包含在 Form 里面，如果没有指明所属对象名称，系统会自动把它当作 Form。

已经是属于 TForm1 的范围了

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    color := clYellow;
end;
```

表 3.2 说明 Delphi 6 所提供的颜色属性：

表 3.2 Delphi 6 所提供的颜色

属性值	意义
clAqua	浅蓝色
clBlack	黑色
clBlue	蓝色
clDkGray	深灰色
clFuchsia	亮紫色
clGray	灰色
clGreen	绿色
clLime	草绿色
clLtGray	浅灰色
clMaroon	栗色
clNavy	深蓝色

续上表

属性值	意义
clOlive	橄榄绿
clPurple	紫色
clRed	红色
clSilver	银色
clTeal	宝蓝色
clWhite	白色
clYellow	黄色
clBackground	目前 windows desktop 的背景颜色
clActiveCaption	使用中窗口的标题栏颜色
clInactiveCaption	非使用中窗口的标题栏颜色
clMenu	菜单的背景颜色
clWindow	目前 windows 的背景颜色
clWindowFrame	目前 windows Frame 的颜色
clMenuText	目前 Menu 的颜色
clWindowText	目前 windows 上文字的颜色
clCaptionText	使用中窗口的标题栏上文字的颜色
clActiveBorder	使用中窗口的边线颜色
clInactiveBorder	非使用中窗口的边线颜色
clAppWorkSpace	应用程序工作区颜色
clHighlight	选择文字(selected text)的背景颜色
clHightlightText	选择文字(selected text)的颜色
clBtnFace	按钮(button)的颜色
clBtnShadow	按钮(button)的阴影颜色
clGrayText	文字被设成无效(dimmed)的颜色
clBtnText	按钮上文字的颜色
clInactiveCaptionText	非使用中窗口的标题栏上文字的颜色
clBtnHighlight	按钮反白的颜色
cl3DDkShadow	按钮阴影颜色(用于 3D 组件)
cl3DLight	用于 3D 组件的 Light color
clInfoText	Tool tip 控制组件的文字颜色
clInfoBk	Tool tip 控制组件的背景颜色

3.2.3　*Font 属性*

谈到这里，我们已经举例说明了与“显示或外表”有关的 BorderStyle 及 Color 两种属性，接着我们要继续介绍与“显示或外表”有关的最后一种属性：Font，以下将用一个实例来说

明 Font 属性的使用方法。

范例 3-4

目的：本范例主要通过修改 TLabel 对象的字号，来说明有关 Font 属性的设置方法。

1. 打开一个新的工程。
2. 在 Form 上加入一个 Label 及一个 Button，如图 3.13 所示。

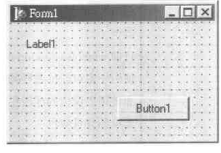


图 3.13　在 Form 上加入一个 Label 及一个 Button

3. 将 Label 的 Font 的大小设为 16、字形样式设为粗斜体、字体设为 Times New Roman 及颜色设为红色，并单击“确定”按钮。如图 3.14 所示。



图 3.14　设计时期改为 Font 的属性

注意

单击“确定”按钮之后，您会发现 Label 的外观跟着改变了。没错，在设计时期改变 Font 的属性就是这么方便。相信您一定会问，那运行时期怎么设置呢？要知道怎么设置，看下去就知道了。

4. 如果您想知道如何在运行时期改变 Font 的属性，请在 Button 的 onClick 事件中填入以下的代码。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    Label1.Font.Size := Label1.Font.Size + 2;
    Label1.Font.Style := [fsBold,fsStrikeOut];
  end;
```

5 运行工程，并按下 Button。运行结果是不是和预期的一样，Label1 的字体大小会随着按下 Button 的次数而变大啊？而且字体也会变为粗体(fsBold)与删除线(fsStrikeOut)。

3.3　与 *Parent* 有关的属性

本小节将为大家说明与 Parent 有关的属性，如果您对 Parent 的意义不是很了解，您可以先阅读第 3.7 节。Parent(父组件)属性可以协助程序开发者管理 Parent 对象与 Chlid 对象的外观属性。如果您将子对象的 ParentColor、ParentFont 及 ParentCtrl3D 属性设为 True，则当您改变父对象的 Color、Font 及 Ctrl3D 等属性时，也会同时改变子对象的相关属性。有一件事您必须要特别注意，如果您将子对象的 ParentColor、ParentFont 及 ParentCtrl3D 属性设为 True 之后，您同样可以改变子对象的 Color、Font 及 Ctrl3D 等属性，但此子对象与父对象有关的属性会自动被设为 False。由于这三种属性的设置方法都一样，因此下面用一个例子来说明。

范例 3-5

目的：本范例主要通过修改 ParentColor 属性来说明与父对象外观属性的设置方法。

1 创建新工程。

2 在 Form 上加入一个 Panel 对象。记录 Panel 的 Color 属性，此时 Color 默认为 clBtnFace。

3 在 Panel 上加入一个 Edit 对象及两个 Button 对象。记录 Edit 的 Color 及 ParentColor 属性，此时 Color 默认为 clWindow、ParentColor 默认为 False。

4 在 Form 上加入一个 ColorDialog 对象(在 Dialog 组件页标签)，如图 3.15 所示。

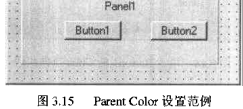


图 3.15　Parent Color 设置范例

5 将 Edit 的 ParentColor 属性改为 True。此时 Edit 的 Color 属性同时被更改为 clBtnFace(因为 Edit 的 Parent 为 Pancel，而 Pancel 的颜色是 clBtnFace)。当然，此时

Edit 的白色背景色同时被改为灰色了。截至目前为止，您应该已经充分了解 ParentColor 的用法了，但这都还是设计时期而已。下一步骤将说明有关 ParentColor 设计时期的用法。

6 在 Button2 的 Click 事件中填入“Edit1.ParentColor:= NOT Edit1.ParentColor;”代码，如图 3.16 所示。

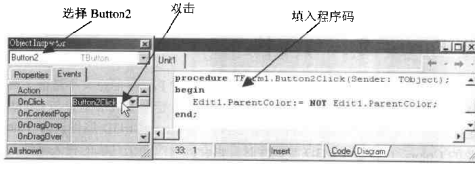


图 3.16　填入 Button2 的 Click 事件代码

注意　这段代码的主要目的是要使 Edit1.ParentColor 的颜色设置，随着 Button2 的 Click 事件而在 True 与 False 两个状态切换，如图 3.17 所示。

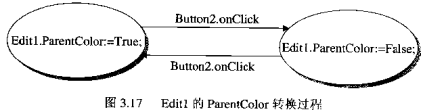


图 3.17　Edit1 的 ParentColor 转换过程

7 请在 Button1 的 Click 事件中填入“if ColorDialog1.Execute then Panel1.Color := ColorDialog1.Color;”代码。如图 3.18 所示。

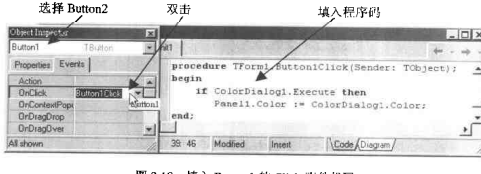


图 3.18　填入 Button1 的 Click 事件代码

注意　这段代码主要是将用户在颜色对话框所选择的颜色，加以设置为 Panel1 的 Color 属性。假如您在对话框没有选择颜色(即按取消)，则 Panel1 的 Color 属性是不会改变的。

8 保存工程。

⑨ 按【F9】功能键运行程序，并按一下 Button1，观察 Panel1 的颜色与 Edit1 的颜色有何改变；再按一下 Button2，并按一下 Button1，再观察 Panel1 的颜色与 Edit1 的颜色有何改变。

3.4　与引导有关的属性

与引导使用有关的属性共有 Caption、TabOrder 及 TabStop 三种。

- Caption 属性

 Caption 属性主要是用来显示标题信息或标识组件文字的字符串，如果您在字符串中的某个字符加上“&”符号，那么就可以配合【Alt】键来选取。

- TabOrder 属性

 TabOrder 属性是用来设置某个了组件在父组件的定位顺序(Tab Order)。换言之，就是用来决定使用【Tab】键的焦点接收次序。TabOrder 属性必须在 TabStop 为 True 时才会有作用。

- TabStop 属性

 TabStop 属性是用来决定该对象的 TabOrder 属性是否有效，换言之，就是决定该对象是否可以接收【Tab】键的焦点。

3.4.1　Caption 属性

Caption 属性主要是用来显示标题信息或标识组件文字的字符串。在此以 Form 为例说明 Caption 在设计时期与运行时期的修改方法。

范例 3-6

目的：本范例的目的主要通过修改 TForm 对象的 Caption 属性，来说明 Caption 属性在设计时期与运行时期的修改方法。

① 创建新工程。

② 将 Form 的 Caption 输入“这是我的 Delphi 6 程序!”，如图 3.19 所示。

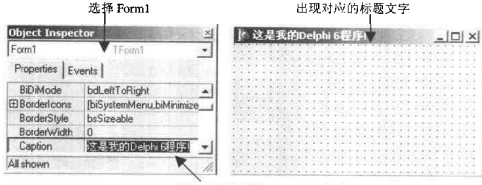


图 3.19　设计时期改变 Form 的 Caption 属性

步骤 3 在 Form 的 Click 事件内容加入“Caption := '现在时间：'+ DateTimeToStr(Now);”。其中 Now 是代表今天的日期及现在的时间，而 DataTimeToStr 是将日期时间的数据类型转为字符串(String)，因为 Caption 是字符串类型的数据，如图 3.20 所示。

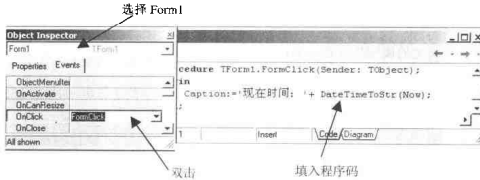


图 3.20 运行时期改变 Form 的 Caption 属性

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程，并在 Form 上按一下鼠标左键，瞧一瞧，在 Form1 的标题栏是不是列出最新的时间啊！

3.4.2 TabOrder 与 TabStop 属性

与引导使用有关的属性除了上述的 Caption 外，尚有 TabOrder 及 TabStop 两种。TabOrder 属性必须在该对象的 TabStop 属性为 True 时才有效，以下用一个例子来说明之。

范例 3-7

目的：本范例通过修改 TabOrder 与 TabStop 属性来说明 TabOrder 与 TabStop 属性的设置及使用方法。

步骤 1 创建工程。

步骤 2 在 Form 上加入两个 TEdit 对象(Edit1 及 Edit2)，放置方式如图 3.21 所示。

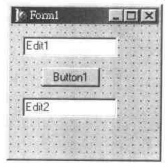


图 3.21 设置 TabOrder 属性

步骤 3　再放置一个 TButton 到 Form 上，其放置方式如图 3.21 所示。此时您可观察 Edit1、Edit2 及 Button1 的 TabOrder 属性，分别为 0、1 及 2。这个顺序是由对象创建的先后次序来决定的，先创建的 TabOrder 较小。

步骤 4　保存工程。

步骤 5　按【F9】功能键运行程序，并重复按下【Tab】键，此时焦点会在 Edit1、Edit2 及 Button1 之间重复出现。您可能会问，这个功能有什么用呢?我使用鼠标不就好了?您说的没错，但是当您鼠标坏了或是在大量文字填写的画面，使用【Tab】键会比鼠标来得快，不信可以自行试试。

步骤 6　如果我们想要改变这个 TabOrder 顺序，您可以直接改变该对象的 TabOrder 属性值，当然您也可以使用更快的方法：在 Form 上单击鼠标右键选择“Tab Order...”，点选所要改变的选项，利用 ↑ 及 ↓ 键来改变次序即可(我们将顺序改变 Edit1、Button1 及 Edit2)，如图 3.22 所示。



图 3.22　利用 Edit Tab Order 来改变 TabOrder 属性

步骤 7　再保存工程及运行程序，再试试【Tab】键，看看结果有何不同。

注意　前面曾经提过 TabOrder 这个属性是否有效，完全由 TabStop 属性值来决定，当 TabStop 为 True 则有效，反之则无效。您现在可以将 Edit2 的 TabStop 设为 False，再将程序重新运行一次，看看结果又有何不同。

3.5　与拖动有关的属性

在 Delphi 6 中控制组件的共同属性除了上述的“与位置或大小有关的属性”、“与显示或外表有关的属性”、“与 Parent 有关的属性”、及“与引导使用相关的属性”外，还有一个非常重要的属性，那就是“对象拖动”。“对象拖动”在未来的用户接口(User Interface)占了相当重要的角色，因为使用“对象拖动”的技术可以让用户随心所欲地调整应用程序的用户接口。例如 Delphi 6 的整合性开发接口、MicroSoft Office 等都是最好的例子。

3.5.1 Drag-and-Drop 自动拖动

表 3.3 将与 Drag-and-Drop 相关的属性与事件列表说明之：

表 3.3 与 Drag-and-Drop 有关的属性名称

属性名称	说明	对象
DragCursor	对象被拖动时鼠标的形状	被拖动对象
DragMode	自动检测或由程控是否对象被拖动	被拖动对象
DragKind	决定对象拖动的类型(这里使用 dkDrag)	被拖动对象

表 3.4 将与 Drag-and-Drop 有关的操作方法列表说明之：

表 3.4 与 Drag-and-Drop 有关的操作方法

操作方法	说明	对象
BeginDrag	对象进入拖动的状态	被拖动对象
Dragging	判断对象是否处于拖动的状态	被拖动对象
EndDrag	停止对象的拖动	被拖动对象

表 3.5 将与 Drag-and-Drop 有关的事件处理列表说明之：

表 3.5 与 Drag-and-Drop 有关的事件处理

事件处理	说明	对象
onDragDrop	当别的对象被移入本对象所触发的事件	被放置对象
onDragOver	当别的对象被移动经过本对象所触发的事件	被放置对象
onEndDrag	对象结束拖动时所触发的事件	被拖动对象
onStratDrag	对象开始拖动时所触发的事件	被拖动对象

要启动对象的拖动状态有自动(DragMode=dmAutomatic)与手动(DragMode=dmManual)两种方法，以下先使用一个例子来说明如何使对象自动的进行拖动。

范例 3-8

目的：本范例主要说明如何使用 Delphi 6 默认的属性来完成对象的拖动。

Step 1 创建新工程(对象摆放位置请参见图 3.23)。

Step 2 将 Form 的 Caption 属性设为“自动拖动范例”。

Step 3 在 Form 上加入两个 Label 对象，其 Caption 属性分别设为“DragOver 事件”与“DragDrop 事件”，并将字体大小设为 10，颜色设为蓝色。

Step 4 在 Form 上加入两个 Edit 对象，Name 属性分别为 Edit1 及 Edit2。

Step 5 在 Form 上加入一个 Button 对象，Name 属性为 Button1、DragCursor 为 crDrag、DragKind 为 dkDrag 及 DragMode 为 dmAutomatic。在这个范例中，此 Button 对象就是被拖动的对象。

步骤 6　在 Form 上加入三个 Panel 对象，Name 属性分别为 Panel1、Panel2 及 Panel3，在 Panel1 的 DragOver 事件内容中加入以下的代码。

```
Edit2.Text:=(Source As TButton).Name+'经过'+(Sender As TPanel).Name;
```

注意　这段代码主要的目的，是在显示 Button 被拖动的过程而已，换言之，它并不会影响程序的运行。由于这个例子总共有三个 Panel 对象，为了要共享代码，因此才使用 As(请参考附录 A 的 Object Pascal)。其中 Source 和 Sender 是 onDragOver 事件的两个参数，Source 代表被移动的对象，也就是 Button1，而 Sender 代表引发 onDragOver 事件的对象(也可以解释为被放置的对象)，也就是 Panel2。如果 Panel1 的 onDragOver 事件内容与 Panel2 及 Panel3 共享后(参考步骤 7)，则 Sender 可能代表 Panel1、Panel2 或 Panel3 等。

步骤 7　将 Panel1 的 DragOver 事件内容与 Panel2 及 Panel3 共享，如图 3.23 所示。

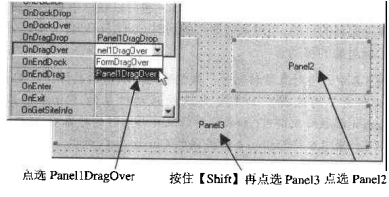


图 3.23　多个对象共享一个事件内容

步骤 8　在 Panel1 的 DragDrop 事件内容中加入以下的代码：

```
procedure TForm1.Panel1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
//显示出 Button 被发到哪个对象上的信息. 其中 Source 是代表
//被移动的对象，而 Sender 代表被放置的对象。
Edit1.Text:=(Source As TButton).Name+'被移到'+ (Sender As TPanel).Name;

//将被拖动对象放置某个对象有两种方法
//方法一、将此 Panel 对象设为 Button1 的 Parent
Button1.Parent:=Sender as TPanel;
Button1.Left:= X;
Button1.Top := Y;

//方法二、并不改变此对象的 Panel 属性，但为了不被其他对
//      象遮住，所以使用 BringToFront 对象方法，以将 Button1
//      放在对象的最上面。
Button1.Left:= Panel1.Left+ X;
Button1.Top := Panel1.Top + Y;
```

```
Button1.BringToFront;
end;
```

STEP 9 并将 Panel1 的 DragDrop 事件内容与 Panel2 及 Panel3 共享，方法参考步骤 7。

STEP 10 在 Form 的 DragOver 事件内容中加入以下的代码：

```
Edit2.Text:= (Source As TButton).Name+'经过'+(Sender As TForm).Name;
```

在 Form 的 DragDrop 事件内容中加入以下的代码：

STEP 11
```
1.Text:=(Source As TButton).Name+'被移到'+(Sender As TForm).Name;
Button1.Left:= X;
Button1.Top := Y;
Button1.Parent:=Form1;
```

STEP 12 保存工程。

STEP 13 按【F9】功能键运行工程。当程序运行时，您可以随意用鼠标左键拖动 Button1，以达到对象搬移的目的。当对象被搬移的同时 Edit1 及 Edit2 会实时显示适当的信息，使您更了解对象的移动。

注意

在范列 3-8 中，当对象被移动时，我们使用了默认的鼠标指针(Cursor)。事实上，您可以有更多的选择。如果要更改被移动对象的 Cursor，只要在该对象的 DragCursor 属性修改即可；如果您要修改非移动时的对象 Cursor，只要修改 Cursor 属性即可。表 3.6 将 Delphi 6 所提供的 Cursor 加以说明。

表 3.6 Cursor 的图标

设置值	设置值	指针图标
crDefault	0	通常都是 crArrow
crNone	-1	没有任何图标
crArrow	-2	
crCross	-3	
crIBeam	-4	
crSizeNESW	-6	
crSizeNS	-7	
CrSizeNWSE	-8	
crSizeWE	-9	
crUpArrow	-10	
crHourGlass	-11	
crDrag	-12	
crNoDrop	-13	
crHSplit	-14	
crVSplit	-15	

续上表

设置值	设置值	指针图标
crMultiDrag	-16	
crSQLWait	-17	
crNo	-18	
crAppStart	-19	
crHelp	-20	
CrHandPoint	-21	
crSize	-22	

3.5.2　*Drag-and-Dock 拖动*

Delphi 6 除了上述对象拖动的方法外，另提供了一种更炫的拖动方法，我们称它为 Drag-and-Dock。表 3.7 将与 Drag-and-Dock 相关的属性与事件列表说明之：

表 3.7　与 Drag-and-Dock 有关的属性名称

名称	说明	对象
DragCursor	对象被拖动时鼠标的形状	被拖动对象
DragMode	自动检测或由程控是否对象被拖放	被拖动对象
DragKind	决定对象拖动的类型(这里使用 dkDock)	被拖动对象
DockSite	决定是否为被放置对象	被放置对象
UseDockManager	决定被放置对象是否使用 DockManager	被放置对象

表 3.8 将与 Drag-and-Dock 有关的事件名称列表说明之：

表 3.8　与 Drag-and-Dock 有关的事件名称

名称	说明	对象
onStartDock	对象开始拖动所触发的事件	被拖动对象
onEndDock	对象结束拖动所触发的事件	被拖动对象
onDockDrop	当别的对象被移入本对象所触发的事件	被放置对象
onDockOver	当别的对象被移动经过本对象所触发的事件	被放置对象

要启动对象的 Drag-and-Dock 状态与 Drag-and-Drop 一样有自动(DragMode)与手动(DragMode=dmManual)两种方法。由于在手动与自动的方法非常类似，因此在手动方面就不再赘述。以下就针对是否有使用 DockManager 来分别说明。



3-9

目的：本范例主要说明 Drag-and-Dock 的自动拖动范例。

STEP 1　创建新的工程(对象摆放位置请参见图 3.24)。

STEP 2　将 Form 的 Caption 属性设为“自动的 Drag-and-Dock 范例”。

3 在Form上加入两个Button对象，其Name属性分别为Button1及Button2。其DragKind属性设为dkDock、DragMode属性设为dmAutomatic。

4 在Form上加入一个Panel对象，其Name属性为Panel1、DockSite属性设为True、UseDockManager属性设为False，如图3.24所示。

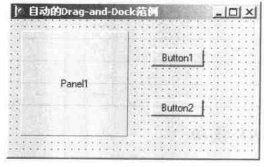


图3.24 未使用Dock Manager的Drag-and-Dock范例

5 保存工程。

6 按【F9】功能键运行工程，工程运行后，您可以将Button1及Button2两个对象轻松的拖动到Panel1对象(见图3.25)；但是，当您再将Button1及Button2两个对象拖动回到Form对象时(因为Form的DockSite属性是被设为False)，则变成一个独立的对象窗口(见图3.26)，这就是Drag-and-Dock与Drag-and-Drop最大的不同之处。

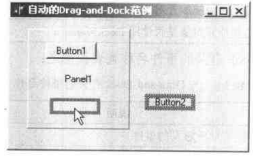


图3.25 未使用Dock Manager的Drag-and-Dock运行结果之一

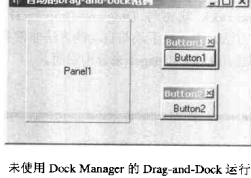


图3.26 未使用Dock Manager的Drag-and-Dock运行结果之二



您也许会觉得很奇怪，为什么在程序一开始运行时，Button1 与 Button2 对象依附在 Form1 上；而从 Panel1 再将 ButtonD1 与 Button2 搬回 Form1 时，它却变成了一个个独立的对象类型，如图 3.26 所示。如果您希望 Button1 与 Button2 可以回到最初的样子，其实只要将 Form1 的 DockSite 属性设为 True 即可。

由上面的范例，您应该已经了解到 Drag-and-Drop 与 Drag-and-Dock 的不同之处。除此之外，Drag-and-Dock 另外提供一个 Dock Manager 管理被 Drag-and-Dock 的对象。简言之，Dock Manager 会自动对被拖动对象加以排列，以下用一个例子来说明。



3-10

由于此范例与范例 3-9 非常相近，因此只要将范例 3-9 步骤 4 中 Panel1 对象的 UseDockManager 属性设为 True 即可(请参考范例 3-9)。保存工程后，请按【F9】功能键运行。当程序运行时，您可以对 Button1 及 Button2 进行拖动。倘若您将 Button1 及 Button2 两个对象都拖动到 Panel1 对象时，其结果如图 3.27 和图 3.28 所示，那是因为 Dock Manager 会根据用户的拖动状况而对被拖动对象进行适当的排列。

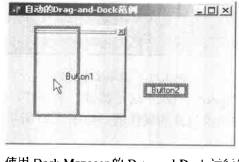


图 3.27　使用 Dock Manager 的 Drag-and-Dock 运行结果之一

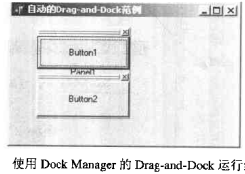


图 3.28　使用 Dock Manager 的 Drag-and-Dock 运行结果之二

3.6　设置共同属性

当在同一个窗体(Form)中有多个具有共同属性的对象，您可以一次将所有共同属性设置为相同的值，而不需要一个一个设置。您只需要选取所要设置的组件，并设置其属性即可，详细过程参考范例 3-11。

范例 3-11

目的：本范例目的主要示范如何同时设置多个对象的相同属性。

(1) 创建新的工程。

(2) 在工程上加入一个 TButton、TLabel 及 TPanel 等对象。

(3) 使用鼠标圈选 TButton、TLabel 及 TPanel 三个对象，如图 3.29 所示。

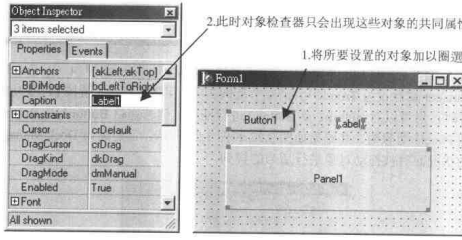


图 3.29　选取多个对象

(4) 在对象检查器的 Caption 加入“这是测试”的字样，此时您会发现这三个对象的 Caption 属性同时被更改为新的属性值，如图 3.30 所示。

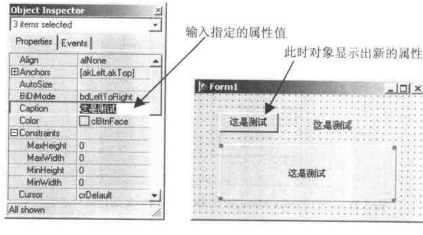


图 3.30　设置共同属性

3.7　*Parent、Owner、Sender 与 Self*

本节的内容主要介绍 Parent、Owner、Sender 与 Self 等四个 Delphi 6 常见的名词，我们

将以范例的方式来做介绍。

3.7.1　*Parent* 与 *Owner*

首先，我们先来谈谈 Parent。您可能会问“什么是 Parent 啊?” Parent 在英文字典中是父亲、母亲或双亲的意思，在 Delphi 6 的世界里，Parent 到底是什么呢? 简单地说，所谓 Parent 就是直接包含本身对象的容器对象。以下用一个例子来说明什么是 Parent。



3-12

目的：本范例主要目的在于说明对象的 Parent 与 Owner 等概念。

1. 创建一个新的工程。
2. 在 Form 上加入一个 GroupBox、一个 Panel。

 并分别在 Form、GroupBox 及 Panel 上各加入一个 Button，如图 3.31 所示。

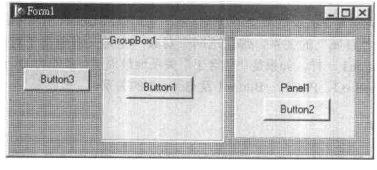


图 3.31　Parent 的属性

在图 3.31 中，您可以发现，当移动 Panel1 组件时，Button2 也会跟着动。同样的，当移动 Groupbox1 组件时，Button1 也会跟着动。像 GroupBox 及 Panel 这种组件就称为容器对象(container)，因此 GroupBox1 就是 Button1 的 Parent、Panel1 也是 Button2 的 Parent。那您一定会问 Button3 的 Parent 是谁呢?测试的方法很简单，您移动到 Form1 上所有的对象(包括 Form1)，看看哪一个对象被移动时 Button3 也会跟着动。答案是 Form1。没错，Form1 也是一个容器对象。表 3.9 就将其间的关系说明之：

表 3.9　Parent 与 Owner

对象名称	Parent	Owner
Form1	Application	TApplication
GroupBox1	Form1	TForm1
Panel1	Form1	TForm1
Button1	GroupBox1	TForm1
Button2	Panel1	TForm1
Button3	Form1	TForm1

从表 3.10 中，您是不是已经了解什么是 Parent 了？但是您一定会问“那什么又是 Owner 呢?”，由于 Owner 与 Parent 比较容易混淆，所以在此一并讨论。所谓 Owner 是指本对象的所有者，以这个例子而言，Form1 的 Owner 及 Parent 都属于 TApplication，而 GroupBox1、Panel1、Button1、Button2 及 Button3 的 Owner 都属于 TForm1。Parent 与 Owner 的概念很重要，因为它会直接影响对象的属性，就像我们在第 3.1 节所讨论的主题一样，不同的 Parent 可能就会直接影响到对象的 Top 或 Left 等属性，如图 3.32 所示。



图 3.32　Object Tree View 与 Parent

如果您对 Owner 与 Parent 的概念不是很了解，您可以从 Object Tree View 窗口中查阅，如果两个对象是“直属”的关系，那么这两个对象一定是父对象与子对象的关系，例如 GroupBox1 与 Button1 一样。如果是“包含于”关系地最顶层对象即为 Owner，例如 Form1 与 Button3、GroupBox1、Panel1、Button1 及 Button2 的关系。

3.7.2　Sender

接着，我们来谈谈什么是 Sender？Sender，我的无敌 CD-67 告诉我它叫“发件人”，但是用“ 发件人”来称呼它是有些奇怪啦，简单地说，Sender 可以称为触发这个事件的组件。下面用一个例子来说明什么是 Sender。

范例 3-13

目的：本范例通过三个 Button 对象，了解 Sender 的意义。

1. 打开一个新的工程。
2. 在 Form1 中加入 Button1、Button2 及 Button3 三个组件。
3. 在 Button1 的 Click 事件填入下述的代码。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if Sender is TButton then
       ShowMessage( (Sender as TButton).Caption )
    else
       ShowMessage('没有Sender');
end;
```

Sender 参数

4 将 Button2 的 onClick 事件处理过程与 Button1 的 onClick 事件处理过程设为共享代码，如图 3.33 所示。

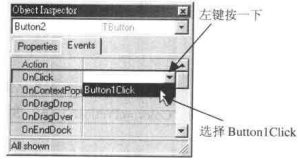


图 3.33　如何多个组件共享一个事件内容

5 在 Button3 的 Click 事件内容中加入以下的代码。

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    Button1Click(nil);
end;
```

6 保存工程。

7 按【F9】功能键运行工程。这个范例共有三种输出，结果如图 3.34 所示。

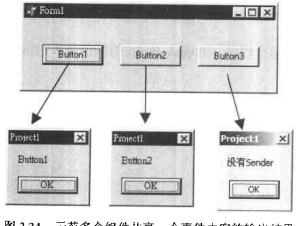


图 3.34　示范多个组件共享一个事件内容的输出结果

由上面的例子您是不是比较了解什么是 Sender 了呢？在 Button1 的 Click 事件使用了一个 nil 的保留字(Reserved words)，那什么又是 nil 呢？nil 就是指没有东西的意思，也就是说，当您通过 Button3 来触发 Button1 的 Click 事件过程中，并没有将 Button3 这个组件送给 Button1 的 Click 事件，这样明白了吗？

注意　以上面的例子而言，如果您希望通过 Button3 去触发 Button1 的 Click 事件的同时，也要将 Button3 送给 Button1 的 Click 事件的话，只要将原来的代码稍微修改即可。

```
procedure TForm1.Button3Click(Sender: TObject);
begin
     Button1Click(Sender);
end;
```

3.7.3 *Self*

接着我们来谈谈什么是 Self？Self 是指目前您编写的代码所属的类别。还记得在范例 3-3 中，我们要改变 Form 的颜色，只写一句“Color:=clYellow;”吗？事实上，我们省略了 Self 这个字。以下用一个例子来说明。



3-14

目的：本范例说明 Self 的意义及使用方法。

1. 创建新的工程。
2. 在 Form.onCreate 的事件中，加入“Self.Color := clYellow;”即可。

```
                    Self 指的是TForm1
procedure TForm1.FormCreate(Sender: TObject);
begin
   self.color:=clYellow;
end;
```

3. 保存工程。
4. 按【F9】功能键运行程序，并确认 Form 的颜色是否和范例 3-3 一样，都是一开始就被改为黄色。

第4章

Delphi 6 对象共同事件

本章目标

面向对象程序设计另一个重要的课题便是对象的事件处理过程。本章先说明如何使用默认的事件处理过程，再从鼠标与键盘两方面来说明 Delphi 6 对象共同的事件处理过程。

本章主题

4.1 使用默认的事件处理过程

4.2 鼠标事件处理过程

4.3 键盘事件处理过程

4.1 使用默认的事件处理过程

在 Delphi 6 的每个对象几乎都会有其默认的事件处理过程。那什么是事件呢?在第 1 章中已经有详细的说明，如果您忘记了，那么请回到第 1 章再复习一遍。以下再一次对事件(Event)及事件处理(Event Handler)做简要的说明。

- 事件(Event)
 事件就像是一个对象的反应行为一样，通常一个对象可以同时产生多个事件。例如：一个 TButton 对象可能有 MouseDown、MouseUp、MouseMove 及 KeyDown 等事件。
- 事件处理(Event Handler)
 事件处理是指被触发的事件所作出的反应代码。在面向对象程序设计中，您所选写的代码都是直接或间接地响应用户所触发的事件。事实上，可以将事件处理程序当作是一个过程(Procedure)。

4.1.1 如何使用默认的事件处理过程

第 1 章曾经提过，一般事件发生原因有“用户事件”、“间接事件”及“系统事件”三种。那么我们应该如何才能得知与使用该对象默认的事件处理过程(Default Event)呢？您只要在对象检查器(Object Inspector)中，快按鼠标左键两下，Delphi 6 便会自动产生所对应的处理过程。产生处理过程后，您的工作就是填入适当的代码即可。

范例 4-1

目的：通过显示现在时间的范例，说明如何使用对象的默认事件处理过程。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TButton 对象，并查看此对象所默认的事件种类。
3. 在 TButton 对象的 onClick 事件处理过程中加入以下的代码。代码中的 ShowMessage 函数是一个经常被使用的信息对话框，其参数值必须为字符串类型的变量；Now 函数可以取得目前系统的日期时间；DateTimeToStr 函数会将指定的日期时间格式转为字符串格式，以利 ShowMessage 函数显示（见图 4.1）。

```
ShowMessage(DateTimeToStr(Now));
```

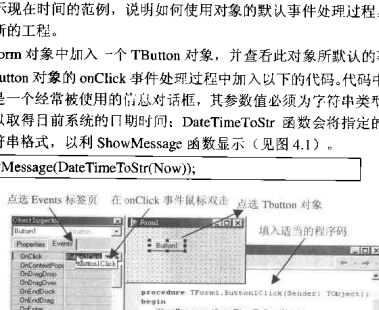


图 4.1 使用默认事件处理过程

4. 保存工程。

5. 按【F9】功能键执行工程。工程执行后，只要触发 Button1 的 Click 事件(即按鼠标左键一下)，事件处理过程便会作出适当的响应，以本范例而言，它会显示出系统现在的日期与时间，如图 4.2 所示。

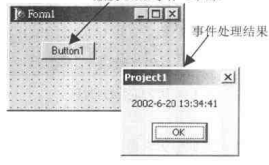


图 4.2　事件与事件处理过程



如果您想删除某个事件处理过程，只需将其事件处理过程内的代码加以删除即可。但如果您想删除某个组件，而这个组件已经存在某些事件处理过程，那么请先删除该组件的所有事件处理过程，再删除该组件。否则，组件是被删除了，但是其事件处理过程的代码却还是存在的。虽然这些代码不会对程序运行造成任何影响，但笔者还是建议将其清除，以利日后程序的维护。

4.1.2　如何编写共享的事件处理过程

在设计程序时，有时候两个对象的事件处理过程可能大同小异，或是完全一样。这个时候您可以写好一份代码，接着再将代码加以复制到其他的事件处理过程中。这样的方法其实在写作的速度上不会太慢，但是这却有一个非常严重的问题隐含在里面。什么严重的问题呢？也许您已经知道了，那就是当程序有更动时，可就惨了，因为您必须同时更改所有“拷贝版”的程序。事实上，您有更好的选择，请参考范例 4-2。



目的：本范例主要目的示范如何让多个对象使用共同的事件处理过程。

1. 创建新的工程。

2. 在工程上加入三个 TButton 对象，其 Name 属性分别为 Button1、Button2 及 Button3。

3. 在 Button1 的 onClick 事件处理过程中加入以下的代码，其中 Sender 代表触发这个事件的对象，as 代表一个类型转换的关键字，TButton 则代表即将被转换的对象类型，而 Name 则是指触发这个事件的对象名称。

```
ShowMessage('您所按下的按钮是 TButton 对象，其名称为' + (Sender as TButton).Name);
```

4 将 Button1 的 onClick 事件处理过程与 Button2 及 Button3 共享，如图 4.3 所示。

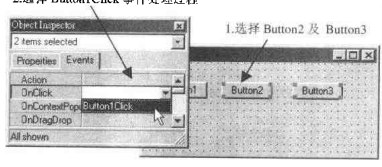


图 4.3 共享事件处理过程

5 保存工程。

6 按【F9】功能键运行工程。此时无论您按下哪一个 TButton，都可以正确地响应此 TButton 对象的名称。因为它们是共享同一个 onClick 事件处理过程，并利用 Sender 参数来判断目前所按下的是哪一个 TButton 对象。

注意 如果您想删除参考其他对象的事件处理过程，而又不想将被参考对象的事件处理过程删除的话，只要在对象检查器中将该对象所参考的事件名称清除即可。如图 4.4 所示。



删除共享事件前　　删除共享事件后

图 4.4 删除共享事件处理过程

注意 在范例中我们使用 as 关键字来进行类型的转换工作，事实上，as 也不太能称为类型转换。因为，当 Sender 与 as 关键字之后所指定的对象类型不符时，程序运行时会发生一个 "Invalid Class Typecast(不合法的类型转换)" 错误。所以如果不确定 Sender 的对象类型，建议您先使用 is 关键字来判断 Sender 所属的对象类型，再使用 as 进行对象类型转换。有关 as 和 is 关键字的详细用法请参考第 5.1.9 节的范例说明。

4.2 鼠标事件处理过程

鼠标是窗口环境里非常重要的输入工具之一。在 Windows 95、Windows 98、Windows NT 4.0 及 Windows 2000 等窗口操作系统中，相信您一定经常使用鼠标作为输入的工具，如果配合上键盘的【Shift】、【Ctrl】及【Alt】等按键更可发挥各种不同的特殊功能。因此如何有效

控制用户的鼠标事件也是窗口程序设计的重要课题，本节将会详细介绍这些相关的功能。

4.2.1　Mouse Down 鼠标事件

一般来说当用户按下鼠标的某一个键之后，便会触发 onMouseDwn 事件。此事件的声明如下(这里以 TButton 对象为例)：

```
procedure Button1MouseDown(Sender: TObject; Button: TMouseButton;
                           Shift: TShiftState; X, Y: Integer);
```

在 onMouseDown 事件中有几个非常重要的参数，如 Sender、Button、Shift、X 及 Y 等。其中 Sender 代表哪个对象触发这个 MouseDown 事件，通常是对象本身，但也会有例外，这个问题已经在第 3.7.2 节与第 4.1.2 节等小节中有详细的说明，在此不再赘述。第二个参数 Button 用来识别用户所按的鼠标键是哪一个(参考表 4.1)，另一个重要参数 Shift 主要是反映出鼠标与键盘之间的关系(参考表 4.2)。而参数 X、Y 代表鼠标按在此对象的坐标。

表 4.1　onMouseDown 事件的 Button 参数值

Button　的参数值	所代表的意义
mbLeft	用户按下鼠标左键
mbMiddle	用户按下鼠标中间键
mbRight	用户按下鼠标右键

表 4.2　onMouseDown 事件的 Shift 参数值

Shift　的参数值	所代表的意义
ssShift	用户按下【Shift】键
ssAlt	用户按下 【Alt】键
ssCtrl	用户按下 【Ctrl】键
ssLeft	用户按下鼠标左键
ssMiddle	用户按下鼠标中间键.
ssRight	用户按下鼠标右键
ssDouble	用户双击鼠标(double-clicked).



4-3

目的：本范例主要说明有关 MouseDown 相关参数的使用方法。

STEP 1　创建新的工程(对象放置位置如图 4.5 所示)。

STEP 2　在 Form 对象上加入一个 TEdit 及两个 TButton 对象。

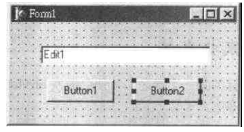


图 4.5　MouseDown 事件范例

3 在 Button1 的 onMouseDown 事件内容中填入下列代码：

```
case Button of
    mbLeft    : Edit1.Text:='您按下左键';
    mbMiddle  : Edit1.Text:='您按下中间键';
    mbRight   : Edit1.Text:='您按下右键';
end;
```

4 在 Button2 的 onMouseDown 事件内容中填入下列代码：

```
if shift = [ssCtrl,ssRight] then
  Edit1.text := '您按下鼠标右键及键盘 Ctrl 功能键';
```

5 保存工程，并按下【F9】功能键运行工程。工程运行后当您在 Button1 对象上按鼠标任意键时，Edit1 会显示您所按下鼠标按键的信息。而您必须在 Button2 对象上同时按下鼠标右键及键盘【Ctrl】功能键才会触发其 onMouseDown 事件。

注意

在范例中的步骤 3 与步骤 4，笔者使用了两个不同的方法来判断 Button 与 Shift 两个参数值。Button 参数是继承自 TMouseButton 枚举类型(Enumerated Type)，而 Shift 参数是继承自 TShiftState 集合类型(Set Type)。所谓“枚举类型”是通过列举出所有可能字符串，而这些字符串是有顺序性的。也就是说您可以利用 ord 函数来取得某个元素在枚举类型中的位置，其顺序值是由 0 开始计算。而“集合类型”是由一些序数类型的元素所组合而成，而且所有的元素必须具有相同的数据类型。因此，“枚举类型”的判断或比较必须使用步骤 3 的方法，而“集合类型”的判断或比较则必须使用步骤 4 的方法。有关 Button 与 Shift 两个参数原始声明如下所示：

```
type TMouseButton = (mbLeft, mbRight, mbMiddle);

type TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight,
                           ssMiddle, ssDouble);
```

4.2.2　*Mouse Move* 鼠标事件

了解 onMouseDown 事件后，接着来谈谈 onMouseMove 事件。一般来说，当用户移动鼠标就会触发此事件，此事件的声明如下(这里以 TForm 对象为例)：

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState;
                               X, Y: Integer);
```

在 onMouseMove 事件中也有几个非常重要的参数，如 Sender、Shift、X 及 Y 等。由于这些参数在 onMouseDown 事件中已经详细说明了，因此不再赘述。接着以一个范例来说明有关 onMouseMove 的用法。

范例 4-4

目的：本范例主要说明有关 MouseMove 相关参数的使用方法。

1. 创建新的工程(对象摆放位置如图 4.6 所示)。
2. 在 TForm 对象上加入两个 TLabel 对象，Name 属性分别为 Label1 及 Label2。将 Label1 的 Caption 属性设为“目前的鼠标坐标：”，并将 Label2 的 Caption 属性设为空字符串。
3. 在 TForm 对象的 onMouseMove 事件内容中加入以下的代码，其主要是通过 IntToStr 转换函数将鼠标所在的位置 X、Y 坐标转为字符串类型加以显示。

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
begin
   Label2.Caption:='X='+IntToStr(X)+',Y='+IntToStr(Y);
end;
```

4. 保存工程。
5. 按【F9】功能键运行程序。当程序运行时只要您移动鼠标，Label2 便会显示出鼠标目前的坐标，结果如图 4.6 所示。

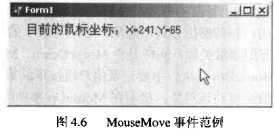


图 4.6　MouseMove 事件范例

4.2.3 *Mouse Up* 鼠标事件

经过了以上的说明，相信您应该对 onMouseDown 及 onMouseMove 事件有相当的了解。接着再来谈谈 onMouseUp 事件。一般来说，当用户放开鼠标按键就会触发此事件，此事件的声明如下(这里以 TForm 对象为例)。

```
procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
                      Shift: TShiftState; X, Y: Integer);
```

在 onMouseUp 事件中同样有几个非常重要的参数，如 Sender、Button、Shift、X 及 Y 等。由于这些参数与 onMouseDown 完全相同，因此不再赘述。

4.2.4 鼠标事件综合练习

通常 MouseUp 事件并不会独自使用，一般来说会与 MouseDown 事件一起使用。以下将用一个较为复杂的范例来作为 onMouseDown、onMouseMove 及 onMouseUp 等鼠标事件的结束。这个范例可以让您在 TForm 上画边框，并且在画完边框之后就产生一个可供移动的真实对象。在这个范例有“画边框”及“移动对象”的两个重要状态概念，第一个“画边框”的状态如图 4.7 所示。

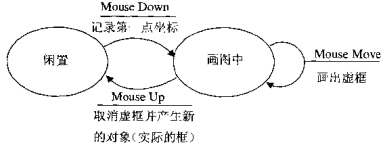


图 4.7 画框的状态图

从上图 4.7 的状态图中可以很清楚的看出，要画一个图必须同时用到 MouseDown、MouseMove 及 MouseUp 三个鼠标事件。在 MouseDown 事件中要记录用户所按下的第一点坐标。在 MouseMove 事件中要画出目前的图形，接着在 MouseUp 事件中要结束绘图的状态。通过上面画框状态图的说明，相信您应该可以想象“移动对象”的状态图会是一个怎么样的图。没错，它和画框的状态图非常类似，同样是由 MouseDown、MouseMove 及 MouseUp 三个鼠标事件所构成的。在 MouseDown 事件中要记录用户所按下的第一点坐标。在 MouseMove 事件中则是根据当初坐标值来移动该对象，接着在 MouseUp 事件中要结束移动对象的状态，状态图请参见图 4.8。

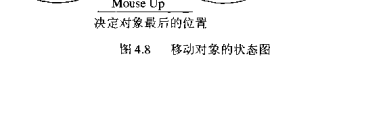


图 4.8 移动对象的状态图

范例 4-5

目的：本范例通过画框与移框的动作让您了解 MouseDown、MouseMove 及 MouseUp 三个鼠标事件的使用方法。在这个范例中，我们利用 TCanvas 对象显示虚框，并利用 TShape 对象代表真实的框，再利用 TList 对象保存所有的 TShape 对象。关于这三个对象的详细说明请参考第 12 章与第 15 章相关小节的详细说明。

步骤 1 创建新的工程。

步骤 2 在 Form 上加入一个 TGroup 对象及八个 TLabel 对象，其说明如表 4.3 所示。

表 4.3

对象(Name　属性)	属性名称	属性设置值
TForm(Form1)	Caption	鼠标事件综合范例
	Color	clWhite
	WindowState	wsMaximized
TGroup(Group1)	Caption	鼠标信息
	Align	alTop
TLabel(Label1)	Caption	Form 上的坐标：
TLabel(Label2)	Caption	Shape 上的坐标：
TLabel(Label3)	Caption	您按下第几个 MyShape：
TLabel(Label4)	Caption	目前共有几个 MyShape：
TLabel(LabelFormXY)	Caption	空字符串
TLabel(LabelShapeXY)	Caption	空字符串
TLabel(LabelWhich)	Caption	空字符串
Tlabel(LabelShapeCount)	Caption	空字符串

步骤 3 在单元中的 Uses 语句中加入 extctrls 单元，因为我们要使用 TShape 对象来当作我们所画的边框，而 TShape 对象被定义在此单元中。

步骤 4 在 Form 对象的私有变量区(Private)加入以下的声明数据：

```
//声明一个 TList 对象，用来装所有的 TShape 对象
  MyList : TList;
   Flag,          //记录 Form 是否进入画图状态的变量
   varX,          //记录画图的第一点坐标
   varY,
   varX2,         //记录画图的第二点坐标
   varY2:Integer;
   ShapeFlag,      //记录 TShape 是否进入移动状态的变量
   varShapeX,     //记录移动 TShape 的第一点坐标
   varShapeY:Integer;
```

```
//自行声明 MouseDown、MouseMove 及 MouseUp 鼠标事件
procedure MyMouseDown(Sender: TObject; Button:
            TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure MyMouseMove(Sender: TObject; Shift:TShiftState; X,Y: Integer);
procedure MyMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState
X, Y: Integer);
```


在这个步骤中，笔者声明三组数字作为记录坐标之用，如 varX、varY、varX1、varY1、varShapeX 及 varShapeY 等；及两个记录状态的变量，如 Flag 与 ShapeFlag。最重要的是，笔者声明了一个 TList 对象，有关这个对象请参考第 15 章的详细介绍。它最主要的目的用于保存程序运行时所产生的“边框”，也就是 TShape 对象。将 TShape 对象保存在 TList 对象中的好处为：可协助程序在运行时顺利找到该对象的位置并移动它。

5 在 TForm 对象的 onCreate 事件内容中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    //设置画笔的颜色及样式
    Canvas.Pen.Color := clRed;
    Canvas.Pen.Style := psDashDot;
    //创建 TList 对象
    MyList:=TList.Create;
end;
```

6 在 TForm 的 onDestroy 事件内容中加入以下的代码，此程序主要是配合上一个步骤而写。在 Delphi 中如果以 Create 操作方法所创建的对象在程序结束前，必须调用 Free 操作方法释放所占用的内存。

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
    //将 MyList 对象加以 Free
MyList.Free;
end;
```

7 在 Form 的 onMouseDown 事件内容中加入以下的代码：

```
procedure TForm1.FormMouseDown(Sender: TObject; Button:
                    TMouseButton;Shift: TShiftState; X, Y: Integer);
begin
    //设置 Form 进入画图状态的指针
    Flag:=1;
    //记录用户所按下的第一点坐标
    varX:=X;
    varY:=Y;
    //将此坐标加以显示给用户看
  LabelFormXY.Caption:='X='+IntToStr(X)+',Y='+IntToStr(Y);
```

```
    //对用来记录用户按下第二点的坐标变量加以初值设置
    varX2:=X;
    varY2:=Y;
end;
```



这个步骤相当于画框的第一个步骤，也就是记录第一点坐标并且将系统状态设为“画图中”。如果您不是很了解，请参见图 4.7 的画框状态图。

步骤 8 在 Form 的 onMouseMove 事件内容中加入以下的代码：

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift:
TShiftState; X,Y: Integer);
begin
    //假如 Form 进入画图状态，则划出虚框图样
    if Flag = 1 then
      begin
        //1.先将原先的图样加以清除
        //1.1 将画笔颜色设为 Form 的 Color
        Canvas.Pen.Color := color;
        //1.2 划出一个方形，参数为第一点及第二点的坐标
        Canvas.Rectangle(varX, varY, varX2, varY2);
        //2.再划出最新的图样，颜色为红色
        canvas.Pen.Color := clRed;
        Canvas.Rectangle(varX, varY, X, Y);
        //3.更新第二点坐标
        varX2:=X;
        varY2:=Y;
        //4.将此坐标加以显示给用户看
        Label1FormXY.Caption:='X='+IntToStr(X)+',Y='+IntToStr(Y);
      end;
end;
```



这个步骤相当于画框的第二个步骤，也就是根据第一点坐标与目前的坐标位置画出一个红色的虚框，目前系统的状态还是处于“画图中”。如果您不了解，请参见图 4.7 的画框状态图。

步骤 9 在 Form 的 onMouseUp 事件内容中加入以下的代码：

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;Shift:
                             TShiftState; X, Y: Integer);
var
      //声明一个 Tshape 的变量
     MyShape : TShape;
begin
  //假如 Form 进入画图状态，则 Create 真正的对象
  if Flag = 1 then
```

```
begin
  //1.将 Form 的画图状态设为 False
  Flag:=0;
  //2.清除原先的图样
  Canvas.Pen.Color := color;
  Canvas.Rectangle(varX, varY, X, Y);
  //3.当用户指定的范围大于 20*20
  if (abs(varX-varX2)>2)) and (abs(varY-varY2:>20) then
    begin
      //3.1 创建一个 TShape 对象
      MyShape := TShape.Create(self);
      //3.2 指定此对象的 Parent 为 Form1，否则无
      //法正确显示
      MyShape.Parent:=Self;
      //3.3 指定此对象与鼠标有关的事件
      MyShape.onMouseDown:-MyMouseDown;
      MyShape.onMouseMove:=MyMouseMove;
      MyShape.onMouseUp:-MyMouseUp;
      //3.4 设置此对象的长、宽
      MyShape.width := Abs(varX - varX2);
      MyShape.Height:- Abs(varY - varY2);
      //3.5 决定此对象的位置，由于使用所划出的坐
      //标有多种状况，故需要加以判断
      if (varX < varX2) and (varY < varY2) then
        begin
          MyShape.Top  := varY;
          MyShape.Left := varX;
        End
      Else if (varX > varX2) and (varY > varY2) then
        begin
          MyShape.Top  := varY2;
          MyShape.Left := varX2;
        end
      else if (varX < varX2) and (varY > varY2) then
        begin
          MyShape.Top  := varY2;
          MyShape.Left := varX;
        end
      else if (varX > varX2) and (varY < varY2) then
        begin
          MyShape.Top  := varY;
          MyShape.Left := varX2;
        end;
```

```
            //3.6 将此对象加入 MyList 中
            MyList.Add(MyShape);
            //3.7 显示目前共有几个 MyShape 的信息
            Label1ShapeCount.Caption:=IntToStr(MyList.Count);
          end;
        //4.利用 Form 对象的 Repaint 操作方法将已经存在的
        //MyShape 对象加以重画(repaint)
        Repaint;
      end;
end;
```



这个步骤相当于画框的最后一个步骤，也就是根据第一点坐标与目前的坐标位置产生一个真实的 TShape 对象。该对象被产生后，系统状态又回到“闲置”。若不了解，请参见图 4.7 的画框状态图。上述程序中，笔者除了动态产生一个 TShape 对象外，并为这个对象指定 onMouseDown、onMouseMove 与 onMouseUp 等事件处理过程。因此，所有被程序动态产生的 TShape 对象都会使用相同的鼠标事件。

STEP 10 在单元中为自定义的 MyMouseDown 事件加入代码：

```
procedure TForm1.MyMouseDown(Sender: TObject; Button:
            TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    //1.显示目前所按下的 MyShape 是第几个
    Label1Which.Caption:=IntToStr(MyList.IndexOf(Sender)+1);
    //2.将此坐标加以显示给用户看
    Label1ShapeXY.Caption:='X='+IntToStr(X)+',Y='+IntToStr(Y);
    //3.设置此对象进入移动状态
    ShapeFlag:=1;
    //4.记录第一点坐标
    varShapeX:=X;
    varShapeY:=Y;
    //5.将此对象带到最前面
    (Sender as TShape).BringToFront;
end;
```



这个步骤相当于移动框的第一个步骤，也就是记录第一点坐标并且将系统状态设为“移动中”。可参见图 4.8 的移动状态图所示。为了显示用户所按下的 TShape 对象是第几个被创建的，因此笔者特别使用 TList 对象的 IndexOf 操作方法找出该对象在 TList 对象中的顺序。除此之外，用户所指定的 TShape 对象可能会被其他的对象覆盖，所以调用 TShape 对象的 BringToFront 操作方法将其显示至最顶层。

STEP 11 在单元中为自定义的 MyMouseMove 事件加入代码：

```
Procedure TForm1.MyMouseMove(Sender: TObject; Shift:
                                    TShiftState; X, Y: Integer);
```

```
Begin
   //假如此对象处于被移动的状态则改变其位置
   if ShapeFlag = 1 then
     begin
       (Sender as TShape).Top := (Sender as Tshape).Top  - Y - varShapeY;
       (Sender as TShape).Left:= (Sender as TShape).Left + X - varShapeX;
     end;
end;
```



这个步骤相当于移动的第二个步骤，也就是将指定的 TShape 对象移动至目前的位置，此时系统的状态还是处于“移动中”。在位置的计算公式中，笔者特别扣除 varShapeY 与 varShapeX 等变量值，其分别代表用户点选此 TShape 对象的第一点坐标，如果没有扣除此坐标值，则会造成显示的不顺畅。可参见图 4.8 的移动状态图。

步骤 12 在单元中为自定义的 MyMouseUp 事件加入代码，这个步骤相当于移动的最后一个步骤，其实只是将系统状态转至“闲置”状态而已。

```
procedure TForm1.MyMouseUp(Sender: TObject; Button:
            TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
   //结束对象的移动状态
   if ShapeFlag = 1 then ShapeFlag:=0;
end;
```

步骤 13 保存工程。

步骤 14 按【F9】功能键运行程序，其结果如图 4.9 所示。

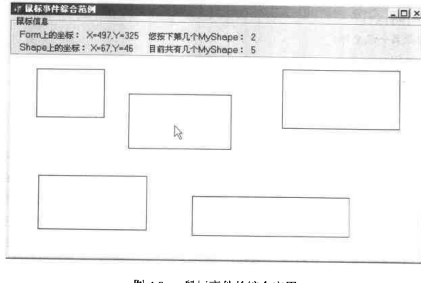


图 4.9　鼠标事件的综合应用

4.2.5　Click 与 DblClick 鼠标事件

在鼠标事件中最常被使用的莫过于 onClick 事件，前面很多例子都已经出现过，本小节将进行更详细的说明。一般来说，onClick 可将它当作是 onMouseDown 及 onMouseUp 两个事件的集合体(以 TForm 为例)。

```
procedure TForm1.FormClick(Sender: TObject);
```

在 onClick 事件中只有一个 Sender 参数，此参数代表是由哪个对象来触发此事件的，通常是对象本身。

倘若将鼠标在某个对象上双击，此时将会触发 onDblClick 事件，onDblClick 可以将它当作是两个 onClick 事件的集合(以 TForm 为例)。

```
procedure TForm1.FormDblClick(Sender: TObject);
```

在 onDblClick 同样只有一个 Sender 参数，此参数亦代表是由哪个对象来触发此事件的，通常是对象本身。以下使用一个例子说明 onClick 及 onDblClick 的用法。



4-6

目的：本范例说明当鼠标单击(Click)及双击(Double Click)时所触发的事件。

步骤 1　创建新工程。

步骤 2　在 Form 对象上加入两个 TLabel 对象，其 Name 属性分别为 Label1 及 Label2。其 Caption 属性分别为“事件运行顺序”及空字符串。

步骤 3　在 Form 对象上加入一个 TButton 对象，将 Caption 属性设为“清除运行结果”，并在此对象的 onClick 事件内容中加入以下的代码：

```
Label2.Caption:='';
```

步骤 4　在 Form 对象的 onMouseDown 事件内容中加入以下代码：

```
Label2.Caption:=Label2.Caption + ' onMouseDown';
```

步骤 5　在 Form 对象的 onMouseUp 事件内容中加入以下的代码：

```
Label2.Caption:=Label2.Caption + ' onMouseUp';
```

步骤 6　在 Form 对象的 onClick 事件内容中加入以下的代码：

```
Label2.Caption:=Label2.Caption + ' onClick';
```

步骤 7　在 Form 对象的 onDblClick 事件内容中加入以下的代码：

```
Label2.Caption:=Label2.Caption + ' onDblClick';
```

⑧ 保存工程。

⑨ 按【F9】功能键运行工程，程序运行后，当您在 TForm 对象上单击，此时 Label2 对象同时显示出与 onClick 有关的事件：onMouseDown、onClick 及 onMouseUp(请参见图 4.10)；当您在 Form 对象上双击，Label2 对象同时显示出与 onDblClick 有关的事件：onMouseDown、onClick 及 onMouseUp、onDblClick、onMouseDown、onMouseUp(请参见图 4.11)。

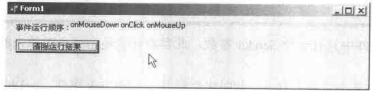


图 4.10　Click 事件的相关事件



图 4.11　DblClick 事件的相关事件

4.2.6　*MouseWheel 鼠标滚轮事件*

MouseWheel、MouseWheel Down 与 MouseWheel Up 等三个鼠标事件是专门为提供滚轮功能的鼠标所设计，换言之，这三个鼠标事件必须是这种特殊的鼠标才能触发它。当用户滚动滚轮时系统便会触发此事件，此事件处理过程的原始声明如下(这里以 TForm 对象为例)：

```
procedure TForm1.FormMouseWheel(Sender: TObject; Shift: TShiftState;
  WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);
```

在 onMouseWheel 事件中有 Sender、Shift、WheelDelta、MousePos 及 Handled 等 5 个参数。Sender 参数代表是由哪个对象来触发此事件的，通常是对象本身；Shift 参数主要是反映出鼠标与键盘之间的关系，请参考表 4.2；WheelDelta 参数代表此次滚轮所转动的次数值；TPoint 参数代表鼠标的坐标位置，TPoint 类型的原始声明如下所示；Handled 参数用来决定此对象是否处理此事件，假如 Handled 值为 False，则表示此事件将由此对象的父对象来处理它。

```
type TPoint = packedrecord
  X: Longint;
  Y: Longint;
end;
```

4.2.7 MouseWheel Down 鼠标事件

当鼠标滚轮被往下转动时会触发 MouseWheel Down 事件。当用户往下滚动滚轮时系统只会触发一次的 MouseWheel 事件，但可能会触发一次或一次以上的 MouseWheel Down 事件，这完全视 MouseWheel 事件中的 WheelDelta 参数而定。MouseWheel Down 事件处理过程的原始声明如下(这里以 TForm 对象为例)：

```
procedure TForm1.FormMouseWheelDown(Sender: TObject;
          Shift: TShiftState;  MousePos: TPoint; var Handled: Boolean);
```

在 onMouseWheelDown 事件中有 Sender、Shift、MousePos 及 Handled 等 4 个参数。Sender 参数代表是由哪个对象来触发此事件的，通常是对象本身；Shift 参数主要是反映鼠标与键盘之间的关系，请参考表 4.2；TPoint 参数代表鼠标的坐标位置；Handled 参数用来决定此对象是否处理此事件，假如 Handled 值为 False，则表示此事件将由此对象的父对象来处理它。

4.2.8 MouseWheel Up 鼠标事件

当鼠标滚轮被往上转动时会触发 MouseWheel Up 事件。当用户往上滚动滚轮时系统只会触发一次的 MouseWheel 事件，但可能会触发一次或一次以上的 MouseWheel Up 事件，这完全视 MouseWheel 事件中的 WheelDelta 参数而定。MouseWheel Up 事件处理过程的原始声明如下(这里以 TForm 对象为例)：

```
procedure TForm1.FormMouseWheelUp(Sender: TObject;
          Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);
```

在 onMouseWheelUp 事件中有 Sender、Shift、MousePos 及 Handled 等 4 个参数。Sender 参数代表是由哪个对象来触发此事件的，通常是对象本身；Shift 参数主要反映出鼠标与键盘之间的关系，请参考表 4.2；TPoint 参数代表鼠标的坐标位置；Handled 参数用来决定此对象是否处理此事件，假如 Handled 值为 False，则表示此事件将由此对象的父对象来处理它。

范例 4-7

目的：本范例说明当用户使用鼠标滚轮时，系统所触发的事件顺序为何。

步骤 1 创建新工程。

步骤 2 在 TForm 对象上摆放 1 个 TMemo 对象。

步骤 3 在 TForm 对象的 onMouseWheel 事件内容中填入以下的代码：

```
Memo1.Lines.Append('[MouseWheel Event] WheelDelta=' +
IntToStr(WheelDelta) +
                    ' MousePos.X=' + IntToStr(MousePos.X) +
                    ' MousePos.Y=' + IntToStr(MousePos.Y)
                    );
```

步骤 4 在 TForm 对象的 onMouseWheelDown 事件内容中填入下列代码：

```
Memo1.Lines.Append('[WheelDown Event]' +
                    ' MousePos.X=' + IntToStr(MousePos.X) +
                    ' MousePos.Y=' + IntToStr(MousePos.Y)
                    );
```

步骤 5 在 TForm 对象的 onMouseWheelUp 事件内容中填入以下的代码：

```
Memo1.Lines.Append('[WheelUp Event]' +
                    ' MousePos.X=' + IntToStr(MousePos.X) +
                    ' MousePos.Y=' + IntToStr(MousePos.Y)
                    );
```

步骤 6 保存工程。

步骤 7 按【F9】功能键运行工程。运行后您可以任意的转动鼠标的滚轮，运行结果如图 4.12 所示。



图 4.12　鼠标的滚轮事件处理范例

通过上面的范例，相信您一定会发现当鼠标的滚轮转动时，可能只会触发一次的 MouseWheel 事件，而可能触发一次或多次的 MouseWheelDown 或 MouseWheelUp 等事件。这完全是由 MouseWheel 事件的 WheelDelta 参数值所决定，该参数值可能为正数或负数。正数代表往上转动，负数则代表往下转动。

4.3　键盘事件

4.3.1　*KeyDown* 键盘事件

与键盘有关的事件有 onKeyDown、onKeyUp 及 onKeyPress 等。在键盘上的按键大致上可分为 ASCII 代码与非 ASCII 代码两类。前者包括 26 个英文字母的大小写及数字，从十进制的 0~255 共有 256 个。后者不属于 ASCII 代码，包括上、下、左、右键及【F1】到【F12】等功能键。不论您按下特殊功能键及一般 ASCII 代码的代码键都会触发这个事件。

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
```

在 onKeyDown 事件内容中有三个重要的参数。其中 Sender 及 Shift 的用法与 onMouseDown 相同，请参考鼠标事件的介绍，在此不再赘述。而 Key 代表所输入的值(数字类型)，以下用一个范例来说明。

范例 4-8

目的：本范例说明当用户按下按键时，系统所接收到的值如何。

步骤 1　包建新工程。

步骤 2　在 Form 对象上摆放 4 个 TLabel 对象，属性如表 4.4 所示。

表 4-4

对象(Name 属性值)	Caption 属性值
TLabel(Label1)	您所按下的按键值是(十进制)
TLabel(Label2)	您所按下的按键值是(十六进制)
TLabel(Label3)	空字符串
TLabel(Label4)	空字符串

步骤 3　在 Form 的 onKeyDown 事件内容中填入以下的代码：

```
Label2.Caption:=IntToStr(Key);
Label4.Caption:=IntToHex(Key,4);
```

步骤 4　保存工程。

步骤 5　按【F9】功能键运行工程。工程运行后，当您按下任意键，程序便会将您所按下的键值显示出来。例如：当我按数字键盘的加号(+)则出现 107(十进制)及 0068(十六进制)的结果，如图 4.13 所示。

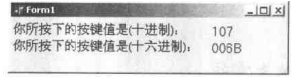


图 4.13　键盘的 onKeyDown 事件范例

通过上面的范例，相信您一定会发现其实在 onKeyDown 的 Key 值是任意键的，而没有大小写之分，表 4.5 列出所有键盘的值供各位参考。

表 4.5　键盘按键值

按键名称	10 进制值	16 进制值
Backspace 键(退格键)	008	0008
Enter 键	013	000D
Shift 键	016	0010
Ctrl 键	017	0011
Alt 键	018	0012
Pause	019	0013
Caps Lock	020	0014
Esc 键	027	001B
空格键	032	0020
Page UP	033	0021
Page Down	034	0022
End 键	035	0023
Home 键	036	0024
方向键(左)	037	0025
方向键(上)	038	0026
方向键(右)	039	0027
方向键(下)	040	0028
Insert 键	045	002D
Delete 键	046	002E
0 键	048	0030
1 键	049	0031
2 键	050	0032
3 键	051	0033
4 键	052	0034
5 键	053	0035
6 键	054	0036
7 键	055	0037
8 键	056	0038

续上表

按键名称	10 进制值	16 进制值
9 键	057	0039
A 键	065	0041
B 键	066	0042
C 键	067	0043
D 键	068	0044
E 键	069	0045
F 键	070	0046
G 键	071	0047
H 键	072	0048
I 键	073	0049
J 键	074	004A
K 键	075	004B
L 键	076	004C
M 键	077	004D
N 键	078	004E
O 键	079	004F
P 键	080	0050
Q 键	081	0051
R 键	082	0052
S 键	083	0053
T 键	084	0054
U 键	085	0055
V 键	086	0056
W 键	087	0057
X 键	088	0058
Y 键	089	0059
Z 键	090	005A
0 (数字键盘)	096	0060
1 (数字键盘)	097	0061
2 (数字键盘)	098	0062
3 (数字键盘)	099	0063
4 (数字键盘)	100	0064
5 (数字键盘)	101	0065
6 (数字键盘)	102	0066
7 (数字键盘)	103	0067

续上表

按键名称	10 进制值	16 进制值
8 (数字键盘)	104	0068
9 (数字键盘)	105	0069
* (数字键盘)	106	006A
+ (数字键盘)	107	0068
- (数字键盘)	109	006D
. (数字键盘)	110	006E
/ (数字键盘)	111	006F
F1 键	112	0070
F2 键	113	0071
F3 键	114	0072
F4 键	115	0073
F5 键	116	0074
F6 键	117	0075
F7 键	118	0076
F8 键	119	0077
F9 键	120	0078
F10 键	121	0079
F11 键	122	007A
F12 键	123	007B
Num Lock	144	0090
Scroll Lock	145	0091
; 键	186	00BA
= 键	187	00BB
, 键	188	00BC
- 键	189	00BD
. 键	190	OOBE
/ 键	191	00BF
~ 键	192	00C0
[键	219	00DB
\ 键	220	00DC
] 键	221	00DD
' 键	222	OODE

4.3.2 *KeyUp* 键盘事件

在键盘事件中另一个事件就是 onKeyUp，当用户放开键盘按键时会触发的事件，它同样

有 Sender、key 及 Shift 等三个重要的参数。由于这些参数与 onKeyDown 事件相同，所以不再重复说明，请参考 onKeyDown 事件。虽然 onKeyUp 事件与 onKeyDown 事件的参数都是相同的，但并不是运行一次 onKeyDown 事件就有一次的 onKeyUp 事件，假如您一直按着某个键不放，onKeyDown 事件会一直连续被触发，而 onKeyUp 事件只会被触发一次。

```
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
```

4.3.3　*KeyPress* 键盘事件

最后将介绍在键盘事件中另一个重要的事件，那就是 onKeyPress 事件。OnKeyPress 与 onKeyDown 最大的不同就是前者只接受 ASCII 键，而 onKeyDown 除接受 ASCII 键外，非 ASCII 键也会接受的。换言之，如果您按住某个 ASCII 键不放，则 onKeyPress 与 onKeyDown 事件会同时被连续触发。

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
```

在 onKeyPress 事件中的 key 与 onKeyDown、onKeyUp 是有所不同的，后者的 key 是一个数字，而前者的 key 是一个字符。由于它是一个字符，因此我们可以在此限定用户所能输入的特定值。



4-9

目的：本范例将示范如何使用 onKeyPress 事件来限定用户输入特定的字符。

1. 创建新的工程。
2. 在 Form 上加入一个 TEdit 对象，在 onKeyPress 事件内容中加入以下的代码：

```
if not (Key in ['0'..'9', '.',#8]) then Key := #0;
```

3. 保存工程。
4. 按【F9】功能键运行工程。由于在 TEdit 对象加入了限制，因此您只能输入 0~9、小数点及“退格键”。

在这个范例中 TEdit 被限制只能输入 0~9、小数点及“退格键”，其中退格键所对应的 ASCII 码为 8，因此#8 就是代表“退格键”。而#0 是代表空字符串。

第5章

TForm 对象

本章目标

TForm 对象是 Delphi 6 面向对象程序设计最主要的对象，几乎大部分的应用程序都以 TForm 对象为基础进行程序的开发。本章将以实例的方式介绍 TForm 对象常用的属性、操作方法及事件处理等。

本章主题

5.1 TForm 对象的属性

5.2 TForm 对象的操作方法

5.3 TForm 对象的事件处理

Delphi 6 程序设计过程中 TForm 对象是大部分程序设计师的重要焦点之一，因为 TForm 对象是创建应用程序的模块单位，只要掌握 TForm 对象的打开、关闭，就能开发出应用程序的流程结构。因此本书特别以一个章节的篇幅来介绍。事实上，到目前为止您已使用过不少的 TForm 对象了，因为在前面两个章节中的每个范例几乎少不了 TForm 对象。

5.1 TForm 对象的属性

TForm 对象有很多的属性，但并非每一个属性都那么常用，因此表 5.1 先将大部分的属性列出来以供参考，但仅介绍一些常用的属性。(注：加下划线的属性代表本章节会有详细说明的属性)

表 5.1 TForm 对象的属性

属性(Property)	▲表示运行时期才能使用	
Action	▲Active	ActiveControl
▲ActiveMDIChild	ActiveOleControl	BorderIcons
BorderStyle	▲Canvas	▲ClientHandle
ClientHeight	▲ClientRect	ClientWidth
DefaultMonitor	Designer	DropTarget
▲Floating	FormStyle	HelpFile
Icon	KeyPreview	▲MDIChildCount
▲MDIChildren	Menu	ModalResult
ObjectMenuItem	OldCreateOrder	OleFormObject
Parent	ParentBiDiMode	PixelsPerInch
Position	PrintScale	Scaaled
TileMode	visible	WindowMenu
WindowsState	AutoScroll	HorzScrollBar
VertScrollBar	BevelEdges	BevelInner
BevelKind	BevelOuter	BevelWidth
BorderWidth	▲Bruch	▲ClientOrigin
▲ControlCount	▲Controls	Ctl3D
▲DockClientCount	▲DockClients	▲DockManager
DockSite	DoubleBuffered	▲Handle
HelpContext	ParentWindow	▲Showing
TabOrder	TabStop	UseDockManager
Align	Anchors	AutoSize
BiDiMode	BoundsRect	Caption
Color	Constraints	ControlState
ControlStyle	Cursor	DockOrientation

续上表

属性(Property)	▲表示运行时期才能使用	
DragKind	DragMode	Enabled
FloatingDockSiteClass	Font	Height
Hint	HostDockSite	Left
Left	LRDockWidth	Name
ParentMenu	ShowHint	TBDockHeight
Top	UnDockHeight	UnDockWidth
Width	WindowsProc	▲ComObject
▲ComponentCount	▲ComponentIndex	▲Components
▲ComponentState	▲ComponentStyle	▲DesignInfo
Owner	Tag	VCLComObject
PopupMenu	Component	DockSite
ParentColor	ParentFont	ParentCtrl3D
DragCursor	AlphaBlend	AlphaBlendValue

在上面所叙述的属性中，事实上您已经认识很多属性了，如与位置有关或大小有关的属性、与显示或外表有关的属性、与 Parent 有关的属性、与引导有关的属性及与拖动(Drag-and-Drop)相关的属性等五大类的属性，属于 Delphi 6 大部分对象的共同属性，用表 5.2 说明，不再详述，详细说明请参考第 3 章。

表 5.2　TForm 对象属性说明

属性	说明
Name	对象的名称，通常是惟一的
Height	对象的高度
Width	对象的宽度
Top	对象的水平位置
Left	对象的垂直位置
BorderStyle	对象的外观类型
Color	对象的颜色
Font	对象的文字颜色
Owner	此对象的所有者
Parent	直接包含此对象的容器对象
ParentColor	继承其 Parent 对象的颜色
ParentFont	继承其 Parent 对象的文字设置
ParentCtrl3D	继承其 Parent 对象的 3D 外观
Ctrl3D	对象的 3D 显示
Caption	对象的标题文字

续上表

属性	说明
DragCursor	对象被拖动时鼠标的形状
Cursor	对象的鼠标形状
DragMode	自动检测或程控对象是否被拖动
DragKind	决定对象拖动的类型
DockSite	决定是否为被放置对象
UseDockManager	决定被放置对象是否使用 DockManager

5.1.1 与活动(Active)相关的属性

Active 属性

Active 属性用来判断该对象是否为 Active。这是一个 Read-only 的属性，您无法去更改它，只能去读取而已。

范例 5-1

目的：本范例说明 TForm 的 Active 属性。

STEP 1 创建新的工程。

STEP 2 在 TForm 的 onClick 事件中写入以下的程序代码：

```
procedure TForm1.FormClick(Sender: TObject);
begin
   if Form1.Active= true then
     Showmessage('Form1 is Active')
   else
     Showmessage('Form1 is not Active');
end;
```

STEP 3 保存工程。

STEP 4 按【F9】功能键运行工程。并在 TForm 对象上 Click 一下，查看目前 TForm 是否处于 Active 的状态。

ActiveControl 属性

ActiveControl 属性用来判断目前在 TForm 对象上是哪一个对象取得焦点。

范例 5-2

目的：本范例使用两个 TButton 对象来说明 ActiveControl 属性。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入两个 TButton 对象，Name 属性分别为 Button1 及 Button2。

STEP 3 在 TForm 对象上加入一个 Timer 对象(在 Sytem 对象页标签上)，并将其 Enabled 属性设为 True，而且在 onTimer 事件内容中填入以下的代码，有关 TTimer 对象请参考第 14.4 节详细说明。

```
Timer1.Interval := 150;
if ActiveControl <> nil then
  ActiveControl.Top := ActiveControl.Top + 5;
```

STEP 4 保存工程。

STEP 5 按【F9】功能键运行工程，此时 Button1 对象会往下自动移动(假如 Button 对象取得焦点的话)，如果您在 Button2 点选一下，则 Button1 对象停止移动，变成 Button2 对象自动移动。

5.1.2　与图标(Icon)相关的属性

BorderIcon 属性

这是一个集合类型的属性，所谓集合类型是指这些属性是具有选择性的。BorderIcon 属性是关于窗口最顶端的控制菜单、最小化按钮及最大化按钮的显示与否，请参考范例 5-3，其可能的值如表 5.3 所示。

表 5.3

属性值	说明
biSystemMenu	显示结束 TForm 的控制按钮
biMinimize	显示最小化按钮
biMaximize	显示最大化按钮
biHelp	显示“?”按钮，必须配合 BorderStyle 为 bsDialog 使用

范例 5-3

目的：本范例将示范如何在运行时期动态设置 BorderIcon 属性值。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入三个 TButton 对象，其相关属性设置如表 5.4 所示。

表 5.4

对象名称	属性	属性值
TButton	Name	btUseAll
	Caption	全部使用
TButton	Name	btCancelOne

续上表

对象名称	属性	属性值
TButton	Caption	取消 项
TButton	Name	btCancelAll
	Caption	全部取消

步骤 3 在 btUseAll 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btUseAllClick(Sender: TObject);
begin
   BorderIcons:=[biSystemMenu,biMinimize,biMaximize];
end;
```

步骤 4 在 btCancelOne 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btCancelOneClick(Sender: TObject);
begin
  BorderIcons := BorderIcons -[biMinimize];
end;
```

步骤 5 在 btCancelAll 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btCancelAllClick(Sender: TObject);
begin
   BorderIcons:=[];
end;
```

步骤 6 保存工程。

步骤 7 按【F9】功能键运行工程，您可以利用三个不同的 TButton 来测试如何在运行时期更改 BorderIcon 属性值。

Icon 属性

在 TForm 对象左上角所显示的图标，默认值为。以下示范如何将此图标更改为新的图标（此图标位于 C:\Program Files\Common Files\Borland Shared\Images\Icons\earth.ico）。

范例 5-4

目的：本范例示范如何将 TForm 对象左上角所显示的图标，更改为新的图标。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象的 Icon 属性值上双击，此时出现“Picture Editor”，如图 5.1 所示。

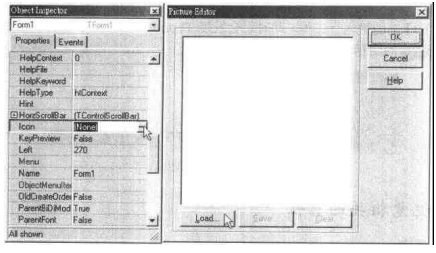


图 5.1　Picture Editor

(3) 在“Picture Editor”中单击 Load... 按钮，此时出现一个“Load Picture”对话框。请在对话框中选取您所要的文件，并单击 打开(O) 按钮。以 C:\Program Files\Common Files\Borland Shared\Images\Icons\earth.ico 为例。

(4) 在“Picture Editor”单击 OK 按钮即可。

(5) 保存工程。

(6) 按【F9】运行工程。此时您会发现窗口左上角的图标已经变成新图标，如图 5.2 所示。

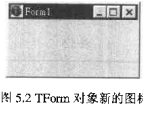


图 5.2 TForm 对象新的图标

注意　上面的方法是针对一个 TForm 对象的 Icon 属性而言，如果您想改变工程的图标，也就是任务栏上或快捷方式上的图标，请参考范例 5-5。

范例 5-5

目的：本范例示范如何将工程的图标，更改为新的图标。

(1) 创建新的工程。

(2) 在菜单栏中选择“Project/Options”，此时出现“Project Options”对话框。

(3) 在“Project Options”对话框中选择“Application”页标签，单击 LoadIcon... 按钮，选择适当的图标，并单击 打开(O) 按钮。以 C:\Program Files\Common Files\Borland Shared\Images\Icons\earth.ico 为例。

(4) 在“Project Options”对话框中选择“Application”页标签，单击 OK 按钮。

5 保存工程。

6 按【F9】功能键运行工程，此时您会发现 TForm 左上角及任务栏上的图标同时变成您所指定的图标了。如图 5.3 所示。

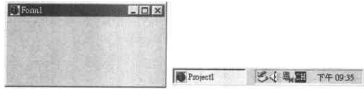


图 5.3 改变工程的图标

5.1.3 与视觉相关的属性

Enabled 属性

决定 TForm 是否有效，若为 False 则 Form 可见，但无法操作；若为 True，则 TForm 对象可取得焦点。例如：Form1.Enabled:=False，表示 Form1 对象被设为无效。

Visible 属性

决定 TForm 是否为可见，若为 True 则为可见，反之，则为不可见。例如：Form1.visible:=False，表示 Form 为不可见。

WindowsState 属性

决定目前 TForm 对象所显示的状态，包含最大化、最小化或为设计时期的大小，如表 5.5 所示。

表 5.5

属性值	说明
wsMaximized	最大化窗口
wsMinimized	最小化窗口
wsNormal	设计时期的大小

FormStyle 属性

决定 TForm 的样式，一般来说 TForm 的样式都是简单的样式，但是如果您想开发一个类似于 Microsoft Word 的应用程序，那您可能会用到 MDI 这个属性值，关于 MDI 的详细说明请参考章节的说明。FormStyle 的属性值如表 5.6 所示。

表 5.6

属性值	说明
fsNormal	一般简单的样式
fsStayOnTop	保持在所有窗口的最前面，不会被其他窗口覆盖
fsMDIForm	MDI 程序的主窗口
fsMDIChild	MDI 程序的子窗口

ShowHint 与 Hint

决定是否显示 Hint 的提示信息，假如 ShowHint 设为 True 时，当鼠标移动到 TForm 上，将触发 onHint 事件，并显示 Hint 字符串。

范例 5-6

创建一个新的工程，并将 TForm 的 ShowHint 属性设为 True，且在 TForm 的 Hint 属性加入提示信息，例如“这是 Hint 属性”，如图 5.4 所示。



图 5.4　Hint 属性

Position 属性

使用 Position 属性可以取得或设置 TForm 对象大小及位置，一般来说 Position 可以有六种属性值，如表 5.7 所示。

表 5.7

属性值	说明
poDesigned	TForm 对象出现在屏幕的位置及大小完全由设计时期决定。
PoDefault	TForm 对象出现在屏幕的位置及大小受分辨率的影响。它每一次运行的位置都会有些稍微的不同，它会由左上角一直往右下角出现。
poDefaultPosOnly	TForm 对象出现在屏幕的位置与 poDefault 设置值相同，但维持设计时期的大小。
poDefaultSizeOnly	这个属性刚好与 poDefaultOnly 相反，TForm 对象出现在屏幕的大小与 poDefault 设置值相同，但维持设计时期的位置。
poDesktopCenter	让 TForm 对象出现在屏幕的中央位置。
poScreenCenter	一般来说此属性值与 poDesktopCenter 相同，惟一的不同只会出现在 multi-monitor applications。

范例 5-7

目的：本工程主要示范如何使用 Postion 属性，首先创建一个新的工程，并将 TForm 对象的 Position 属性设为 poDefault。重复运行工程，此时您会发现 TForm 对象每次出现的位置都会有些不同，如图 5.5 所示。

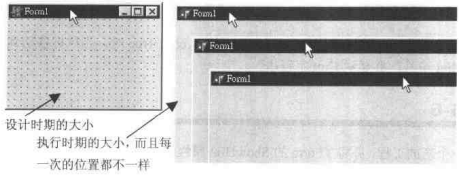


图 5.5 Position 属性(以 poDefault 为例)

ClientHeigh 属性与 ClientWidth 属性

ClientHeight 属性与 ClientWidth 属性分别代表 TForm 对象的工作区高度及宽度，这里的高度并不包含标题栏。

ClientRect 属性

ClientRect 属性可以当成是 ClientHeight 及 ClientWidth 的集合体。ClientRect 继承自 TRect 对象。而 TRect 对象可能会有两种类型出现：一种是数字类型，而另一种是 TPoint 数据类型。

```
TPoint = record
  X: Longint;
  Y: Longint;
end;
TRect = record
  case Integer of
    0: (Left, Top, Right, Bottom: Integer);
    1: (TopLeft, BottomRight: TPoint);
end;
```

换句话说，ClientRect 也就等于 Rect(0,0,ClientWidth, Client Height)。

范例 5-8

1. 创建新的工程。
2. 在 TForm 对象上加入两个 TLabel 对象，其 Caption 属性分别设为“TForm 的 ClientWidth 值”及“TForm 的 ClientHeight 值”。
3. 在 TForm 对象上加入两个 TSpinEdit 对象(在 Samples 对象页标签中)，其 Name 属性分别为 SpinEdit1 及 SpinEdit2，分别代表 TForm 的 ClientWidth 值及 TForm 的 ClientHeight 值，如图 5.6 所示。



图 5.6　TForm 对象的 ClientWidth 及 ClientHeight

4. 在 TForm 对象的 onCreate 事件内容中加入以下的代码。其目的是将 TForm 对象的 ClientWidth 及 ClientHeight 的值设为 SpinEdit1 及 SpinEdit2 的初值：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   SpinEdit1.value:=ClientWidth;
   SpinEdit2.value:=ClientHeight;
end;
```

5. 在 SpinEdit1 对象的 onChange 事件内容中加入以下代码。其目的是使 TForm 对象的 ClientWidth 随着 SpinEdit1 的 Value 值而改变：

```
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
   ClientWidth:=SpinEdit1.Value;
end;
```

6. 在 SpinEdit2 对象的 onChange 事件内容中加入以下代码。其目的是使 TForm 对象的 ClientHeight 随着 SpinEdit2 的 Value 值而改变：

```
procedure TForm1.SpinEdit2Change(Sender: TObject);
begin
   ClientHeight:=SpinEdit2.Value;
end;
```

7. 保存工程。

8. 按【F9】功能键运行工程，工程运行后您可以利用 TSpanEdit 对象来动态改变 TForm 对象的 ClientWidth 属性与 ClientHeight 属性。

Align 属性

Align 属性是用来设置 TForm 对象在运行时的位置，很多对象也都具有这个属性，属性 Align 的值共可分为六种，如表 5.8 所示。

表 5.8

Align 属性值	说明
alNone	不改变 TForm 对象的位置
alTop	将 TForm 对象移到屏幕的最上方，保留原本的高度值(Height)
alBottom	将 TForm 对象移到屏幕的最下方，保留原本的高度值(Height)
alLeft	将 TForm 对象移到屏幕的最左方，保留原本的宽度值(Width)

续上表

Align 属性值	说明
alRight	将 TForm 对象移到屏幕的最右方，保留原本的宽度值(Width)
alClient	将 TForm 对象填满整个屏幕

范例 5-9

目的：本范例说明 TForm 对象的 Align 属性。

1. 创建新的工程。
2. 在 TForm 对象上加入 6 个 TButton 对象，其属性与 onClick 事件内容分别如表 5.9 所示。

表 5.9

Name 及 Caption 属性值	OnClick 事件内容
Button1(alBottom)	Align:=alBottom;
Button2(alTop)	Align:=alTop;
Button3(alRight)	Align:=alRight;
Button4(alLeft)	Align:=alLeft;
Button5(alNone)	Align:=alNone; Width:=351; Height:=141;
Button6(alClient)	Align:=alClient;

3. 保存工程。
4. 运行工程，以 alTop 为例，其运行结果如图 5.7 所示。

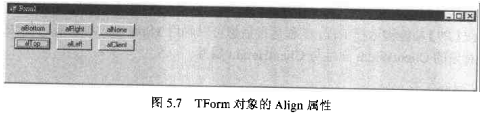


图 5.7 TForm 对象的 Align 属性

AlphaBlend 属性

AlphaBlend 属性用来决定 TForm 对象是否支持透明的显示方式，这个属性必须在 Widnows 2000 或以上版本的操作系统，而且 CPU 必须在 Pentium 90 或以上的才会生效。当 AlphaBlend 属性值为 True 表示支持，反之，则不支持。另外，AlphaBlend 属性必须配合 AlphaBlendValue 属性使用。

AlphaBlendValue 属性

AlphaBlendValue 属性用来决定 TForm 对象支持透明显示方法的透明程度，其属性值介

于 0~255 之间，但必须配合 AlphaBlend 属性一起使用。当属性值为 255 时代表完全不透明，反之，当属性值为 0 时，代表完全透明。

TransparentColor 属性

TranspartentColor 属性可决定是否将 TForm 对象的指定颜色设为透明，这个属性必须在 Widnows 2000 或以上版本的操作系统，而且 CPU 必须在 Pentium 90 或以上的才会生效。当 TranspartentColor 属性值为 True 表示支持，反之，则不支持。另外，TranspartentColor 属性必须配合 TransparentColorValue 属性使用。

TransparentColorValue 属性

TransparentColorValue 属性用来决定 TForm 对象支持透明显示的颜色，其属性值为 TColor 对象，但必须配合 TranspartentColor 属性一起使用。

5.1.4　与引导相关的属性

TabOrder 属性

TabOrder 属性是指当用户按【Tab】键，控制组件取得焦点的顺序，顺序值通常是由 0 开始计算。当然您也可以利用 Edit Tab Order 来改变 TabOrder 属性，请参考范例 3-7。

TabStop 属性

TabStop 属性用来决定用户按下【Tab】键时，该对象是否可以接受焦点，参考范例 3-7。

5.1.5　与绘图相关的属性

Canvas 属性

Canvas 属性是一个运行时期才能使用的属性，继承自 TCanvas 对象。换句话说，TCanvas 对象是一张附属在 TForm 对象上的透明画纸，您可以在上面任意的画图，或加载图片文件。有关 Tcanvas 对象请参考相关章节的详细说明。

5.1.6　与菜单相关的属性

PopupMenu 属性

PopupMenu 属性通常配合 TPopupMenu 对象一起使用。图形化用户接口有一个很特别的对象就是 TPopupMenu，如果您使用 Windows 95/98/2000 有一段时间，您一定知道 PopupMenu 的好处，因此您可以在工程中加入适当的 PopupMenu 以增加程序的易用性。以下将简单地说明如何为 TForm 对象加入 TPopupMenu 对象，有关 TPopupMenu 对象请参考相关章节的详细说明。

范例 5-10

目的：本范例示范如何使用 PopupMenu 属性。

1. 创建新的工程。
2. 在 TForm 对象上加入一个 TPopupMenu 对象，并将该对象的 Name 属性设置为 PopupMenu1。
3. 点一下 PopupMenu1 的 Items 属性右边的 按钮，此时 Delphi 6 会打开“Form1.PopupMenu1”对话框。并分别在 Capiton 属性中输入“红色”、“白色”及“黑色”等三个 Item，如图 5.8 所示。

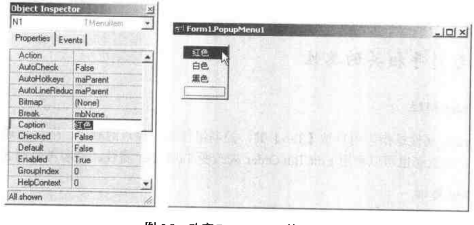


图 5.8 改变 Popup Menu 的 Items

4. 在 PopupMenu1 的“红色”、“白色”及“黑色”等三个 Items 中的 onClick 事件内容中分别加入代码，如表 5.10 所示。

表 5.10

选项	OnClick 事件的代码
红色	Color:=clRed;
白色	Color:=clWhite;
黑色	Color:=clBlack;

5. 最重要的一件事，就是别忘了在 TForm 的 PopupMenu 属性选择 PopupMenu1 对象。
6. 保存工程。
7. 按【F9】运行工程，程序运行后您可以使用鼠标右键，动态地改变 TForm 对象的颜色，如图 5.9 所示。

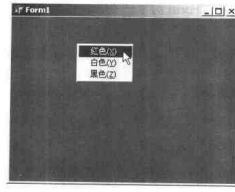


图 5.9　Popup 属性与 TPopupMenu 对象

Menu 属性

如果在工程中想要提供用户很多功能选项，那么 Menu 属性您一定会用到，一般来说 Menu 属性是配合 TMainMenu 对象使用，有关 TMainMenu 对象请参考相关章节的说明。以下范例将示范如何利用 TMainMenu 对象来替代在范例 5-10 中 TPopupMenu 所扮演的角色。

范例 5-11

目的：本范例示范如何使用 Menu 属性。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象上加入一个 TmainMenu 对象，并将该对象的 Name 属性设置为 MainMenu1。

步骤 3　在 MainMenu1 的 Items 属性上双击，此时出现“Form1. MainMenu1”对话框。并分别在 Capiton 属性中输入“红色”、“白色”及“黑色”等三个 Item，如图 5.10 所示。

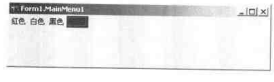


图 5.10　在 TmainMenu 对象中加入适当的选项

步骤 4　在 MainMenu1 的“红色”、“白色”及“黑色”等三个 Items 中的 onClick 事件内容中分别加入代码，如表 5.11 所示。

表 5.11

选项	OnClick 事件的代码
红色	Color:=clRed;
白色	Color:=clWhite;
黑色	Color:=clBlack;

步骤 5 最重要的是别忘了要将 TForm 对象的 Menu 属性值设为“MainMenu1”。

步骤 6 保存工程。

步骤 7 运行工程，程序运行后，您可以利用 MainMenu 来动态改变 TForm 的颜色。

5.1.7 与键盘相关的属性

KeyPreview 属性

允许 TForm 对象在其控制组件之前接收有关键盘的信息，如果设置 KeyPreview:=True 表示可以接收键盘信息，反之则不接收。

范例 5-12

目的：本范例说明 KeyPreview 属性的用法。

步骤 1 创建新的工程。

步骤 2 将 TForm 对象的 KeyPreview 属性设为 True。

步骤 3 在工程的变量声明区加入以下的代码：

```
var
  //声明一个 TColor 的变量，主要用来记录 TForm 原来的颜色
  FormColor:TColor;
```

步骤 4 在 TForm 对象的 onKeyDown 事件内容中加入以下代码：

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key:
                                Word; Shift: TShiftState);
begin
  //记录 TForm 原来的颜色
  FormColor := Color;
  //设置新的颜色给 TForm
  Color := clWhite;
end;
```

步骤 5 在 TForm 对象的 onKeyUp 事件内容中加入以下代码：

```
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
                                              Shift: TShiftState);
begin
  //将 TForm 的颜色恢复回原来的颜色
  Color := FormColor;
end;
```

步骤 6 保存工程。

步骤 7 运行工程，程序运行后，当您按下键盘任意按键时，Form 对象颜色均会变成白色，而当您放开该按键时，Form 对象又会恢复原来的颜色。至于如何取得用户所按下的

按键，请参考范例 4-8。



运行范例 5-12 时您可能会发现，当您按住某个按键不放或按久一点，那么 TForm 对象的颜色就变不回来了。为什么呢?我们在第 4.3.2 节中曾经说过，假如您一直按着某个键不放，那么 onKeyDown 事件会一直连续被触发，而 onKeyUp 事件只会被触发一次。您现在知道这个范例的 bug 在哪里了吧!

5.1.8　与滚动条(ScrollBar)相关的属性

AutoScroll、HorzScrollBar 与 VertScrollBar 属性

AutoScroll 属性是表示当 TForm 上的对象超出范围大于 TForm 范围时，是否要出现 ScrollBar。当 AutoScroll:=True 表示会自动出现，反之，则不会自动出现，参考范例 5-13。HorzScrollBar 及 VertScrollBar 属性分别用来设置水平与垂直的 ScrollBar，当 ScrollBar 的 Range 大于 TForm 所对应的 ClientHeight 或 ClientWidth 才会有效，参考范例 5-14。HorzScrollBar 及 VertScrollBar 的相关属性介绍如表 5.12 所示。

表 5.12

属性	说明
ButtonSize	两端◀及▶按钮的大小
Size	ScrollBar 的高度(以 HorzScrollBar 为例)
Position	目前滚动所到的位置
Color	ScrollBar 的背景颜色
Increment	ScrollBar 微调的大小
Range	是 ScrollBar 所包含的范围，通常要大于 TForm 的 ClientRec，ScrollBar 才会出现
Style	ScrollBar 的类型，有 ssRegular、ssHotTrack 及 ssFlat 三种
Smooth	决定 ScrollBar 是否为平滑的对象外观
Visible	决定 ScrollBar 是否有可见对象

范例 5-13

目的：本范例示范如何使用 AutoScroll 属性。

步骤 1　创建一个新的工程。

步骤 2　将 TForm 的 AutoScroll 属性设为 True。

步骤 3　加入一个 TButton，Capiton 属性设置“AutoScroll”，在 onClick 事件内容加“AutoScroll:=NOT AutoScroll;”代码，此时 TForm 会自动出现 ScrollBar，如图 5.11 所示。

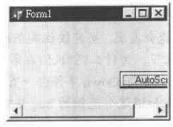


图 5.11 TForm 对象的 AutoScroll 属性

4 保存工程。

5 运行工程，此时您可以发现当 AutoScroll 被设为 False 时，就算有对象超出 TForm 的范围，也不会出现 ScrollBar。

范例 5-14

目的：本范例示范如何使用 HorzScrollBar 属性。

1 创建新的工程。

2 在 TForm 对象的 onCreate 事件内容中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //设置 TForm 的 ClientWidth
  ClientWidth := 250;
  //设置水平的 ScrollBar
  with HorzScrollBar do
  begin
    //设置 ScrollBar 的宽度，必须要比 ClientWidth 大才会出现
    Range := 400;
    //目前滚动所到的位置
    Position := Range - ClientWidth;
    //微调的大小
    Increment := 10;
    Visible := True;
  end;
end;
```

3 保存工程。

4 按【F9】运行工程，程序运行时，您会发现在 TForm 对象上多出一个水平的 ScrollBar，如图 5.12 所示。

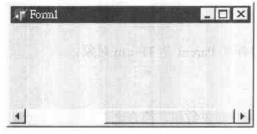


图 5.12 TForm 对象的 HorzScrollBar 属性

AutoSize 属性

当 AutoSize 属性值为 True 时，Delphi 6 会根据目前在 TForm 上的所有对象来决定 TForm 对象的大小，如图 5.13 所示。

范例 5-15

创建新的工程，并且在 TForm 上加入一个 TButton 对象，将 TButton 对象的 Caption 属性设为“AutoSize”，并在其 onClick 事件内容中加入“AutoSize:=True;”的代码。程序的运行结果如图 5.13 所示。

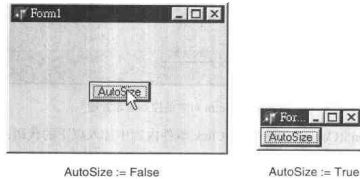


图 5.13　TForm 对象的 AutoSize 属性

5.1.9　与控制组件相关的属性

ComponentCount 属性

ComponentCount 属性是指目前有多少个控制对象，换言之，只要某个对象的 Owner 是 TForm 那就算是 TForm 对象的控制对象。

Components 属性

是指目前 TForm 上的控制对象数组。换言之，只要是 Owner 的 TForm 对象均会存在此数组。

- *ControlCount* 属性

 是指目前有多少控制组件的 Parent 是 TForm 对象。

- *Controls* 属性

 是指目前 TForm 上的控制对象数组。换言之，只要是 Parent 的 TForm 对象均会存在此数组。

范例 5-16

目的： 本范例说明 TForm 对象中与控制组件相关属性的应用。

1. 创建新的工程。
2. 在 TForm 对象中加入 6 个 TButton 及一个 TPanel 对象，将 Button5 及 Button6 对象的 Caption 属性设置“Components”及“Controls”，其摆放位置如图 5.14 所示。

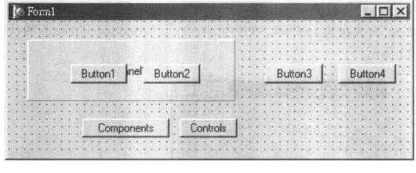


图 5.14 TForm 对象的控制对象管理

3. 请在 Button5(Components)的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.Button5Click(Sender: TObject);
var
    //声明两个变量
    I:Integer;
    MyString:String;
begin
    //设置 MyString 变量的初值，并加入换行符号
    MyString:='以下对象的 Owner 都是 TForm 对象: '+#10;
    //利用 ComponentCount 取得控制组件的个数
    For I:=0 to ComponentCount -1 do
        //判断此对象是 TButton 或 TPanel 对象，并将记
        //录此对象的 Name 属性
        if Components[I] is TButton then
          MyString:=MyString + TButton(Components[I]).Name + #10
        else if Components[I] is TPanel then
```

```
            MyString:=MyString + TPanel(Components[I]).Name + 10;
      //show 出这些对象的 Name 属性
      ShowMessage(MyString);
  end;
```

步骤 4 在 Button6(Controls)的 onClick 事件内容中加入以下的代码。

```
  procedure TForm1.Button6Click(Sender: TObject);
  var    //声明两个变量
     I:Integer;
     MyString:String;
  Begin
     //设置 MyString 变量的初值，并加入换行符号
     MyString:='以下对象的 Parent 都是 TForm 对象: '+#10;
     //利用 ControlCount 取得控制组件的个数
     For I:=0 to ControlCount -1 do
        //判断此对象是 TButton 或 TPanel 对象，并将记录
        //此对象的 Name 属性
        if Controls[I] is TButton then
           MyString:=MyString+ TButton(Controls[I]).Name + #10
        else if Components[I] is TPanel then
           MyString:=MyString+ TPanel(Controls[I]).Name + #10;
     //show 出这些对象的 Name 属性
     ShowMessage(MyString);
  end;
```

步骤 5 保存工程。

步骤 6 运行工程。当您按下 Button5 对象时，程序会显示 Owner 为 TForm 的对象名称。按下 Button6 对象时，程序会显示 Parent 为 TForm 的对象名称，如图 5.15 所示。

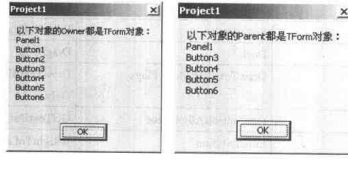


图 5.15　运行时期取得 TForm 的控制对象

注意　ASCII 10 也就是程序中所使用的#10，我们也称它为 Linefeed 字符；另外，ASCII 13(#13)我们也称它为 Carriage Return 字符。从现在开始，您可以单独使用 ASCII 10 来取代以前#10#13 两个字符。

5.2 TForm 对象的操作方法

TForm 对象有很多的操作方法，但并非每一个操作方法都常用，因此表 5.13 将所有的操作方法列出来以供参考，但仅介绍一些常用的操作方法(注：加下划线的属性代表本章节有详细介绍的操作方法)。

表 5.13 TForm 对象的操作方法

操作方法(Method)		
ArrangeIcons	Cascade	Next
Previous	Tile	AfterConstruction
BeforeDestruction	Close	CloseQuery
Create	CreateNew	DefocusControl
Destroy	FocusControl	GetFormImage
Hide	IsShortCut	MouseWheelHandler
Print	Release	SendCancelMode
SetFocus	SetFocusedControl	Show
ShowModal	WantChildKey	DisableAutoRange
EnableAutoRange	ScrollInView	Broadcast
CanFocus	ContainsControl	ControlAtPos
CreateParented	CreateParented Control	DisableAlign
DockDrop	EnableAlign	FindChildControl
FlipChildren	Focused	GetTabOrderList
HandleAllocated	HandleNeeded	InsertControl
Invalidate	PaintTo	Realign
RemoveControl	Repaint	ScaleBy
ScrollBy	SetBounds	Update
UpdateControlState	BeginDrag	BringToFront
ClientToScreen	Dock	DragDrop
Dragging	DrawTextBiDiMode Flags	DrawTextBiDiModeFlagsReading Only
EndDrag	GetControlsAlign ment	GetTextBuf
GetTextLen	InitiateAction	IsRightToLeft
ManualDock	ManualFloat	Perform
Refresh	ReplaceDocked Control	ScreenToClient
SendToBack	SetTextBuf	UseRightToLeftAlignment
UseRightToLeft Reading	UseRightToLeftScroll Bar	DestroyComponents
Destroying	ExecuteAction	FindComponent
FreeNotification	FreeOnRelease	GetNamePath

续上表

操作方法(Method)		
GetParentComponent	HasParent	InsertComponent
RemoveComponent	SafeCallException	UpdateAction
Assign	ClassInfo	ClassName
ClassNameIs	ClassParent	ClassType
CleanupInstance	DefaultHandler	Dispatch
FieldAddress	Free	FreeInstance
GetInterface	GetInterfaceEntry	GetInterfaceTable
InheritsFrom	InitInstance	InstanceSize
MethodAddress	MethodName	NewInstance

5.2.1　TForm 对象取得与释放

Create 操作方法

Create 是每一个对象共有的操作方法(Method)，Create 就是真正产生一个对象。还记得范例 4-5 的例子吗?在那个范例中我们已经使用了 Create 的 Method，虽然在范例中是用来产生 TShape 对象，但事实上方法是差不多的。Create 操作方法的原始类型如下：

```
constructor Create(AOwner: TComponent); override;
```

Free 操作方法

俗语说的好“有借有还，再借不难”，如果 Create 是向系统借放，那么 Free 便是将款项还给系统。Free 操作方法通常用于毁坏(Destroy)动态产生的 TForm 对象，并且释放(Free)其相关的内存。如果您使用 Free 操作方法来释放对象时，假如此对象实例(Object Instance)不为 nil，此时 Free 操作方法会自动调用 Destroy 操作方法来释放内存中的对象。因此任何动态产生(或是没有指定 Owner 的对象)的对象都可使用 Free 操作方法来释放，Free 操作方法的原始类型如下：

```
procedure Free;
```

Destroy 操作方法

如果您想真正的从内存里将 TForm 对象删除，那么您必须使用 Destroy 的操作方法，当用户使用 Destroy 操作方法时，假如该对象不为空值(nil)时，系统会触发 onDestroy 事件，接着进行内存释放的工作。我们在实务上很少直接调用 Destroy 操作方法，而是调用 Free 操作方法来达到释放的目的，Destroy 操作方法的原始类型如下：

```
destructor Destroy; override;
```

Close 操作方法

如果您要结束 TForm 对象，可以使用 Close 操作方法。当 Close 操作方法被调用时，系统会触发 onCloseQuery 事件，并通过 CanClose 参数来判断此 TForm 对象是否可以被关闭。假如 CanClose 参数为 True，则表示此 TForm 对象是可以被关闭的，因此 onClose 事件会被触发。假如 CanClose 参数为 False，则 Close 操作方法将会被中断。Close 操作方法的原始类型如下：

```
procedure Close;
```

CloseQuery 操作方法

CloseQuery 操作方法与 Close 操作方法类似，其同样会触发 onCloseQuery 事件。此两者操作方法惟一不同的在于前者只能获取 onCloseQuery 事件的 CanClose 参数值，而无法真正的关闭 TForm 对象。当 CloseQuery 操作方法被调用时，CloseQuery 操作方法会返回 Boolean 值，当返回值为 True，则表示此 TForm 对象是可以被关闭的，反之，是无法被关闭的。CloseQuery 操作方法的原始类型如下：

```
function CloseQuery: Boolean; virtual;
```

范例 5-17

目的：本范例示范如何创建新的 TForm 对象，并介绍 TForm 对象之间的切换方法。

步骤 1 创建新的工程。

步骤 2 在 TForm1 对象中加入一个 TButton 对象与一个 TMemo 对象，其相关属性设置如表 5.14 所示。

表 5.14

对象名称	属性名称	属性值
TButton	Caption	Show Form2
TMemo	Lines	(空白)

步骤 3 选择菜单栏中的“File / New Form”，产生一个新的 TForm 对象，Name 属性设为 Form2。并将此 Form2 设为 Available form，而非 Auto-Create form。其设置步骤请参考第 2.2.2 节。

步骤 4 并且在 Form2 对象上加入一个 TButton 对象，且将其 Capiton 属性设置为“Close Form2”，并在此 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  //调用 Close 操作方法
  Form1.Memo1.lines.add('调用 Close 操作方法');
```

```
  Close;
end;
```

STEP 5 本范例主要是为了让您了解有关 Create、Free、Destroy、Close 及 CloseQuery 等操作方法的运行顺序，因此请依序在这些操作方法的相关事件中加入适当的代码。首先，请在 Form2 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm2.FormCreate(Sender: TObject);
begin
  //显示事件名称
  Form1.Memo1.lines.add('现在触发 onCreate 事件!!');
end;
```

STEP 6 请在 Form2 对象的 onDestroy 事件处理过程中加入以下的代码：

```
procedure TForm2.FormDestroy(Sender: TObject);
begin
  //显示事件名称
  Form1.Memo1.lines.add('现在触发 onDestroy 事件!!');
end;
```

STEP 7 请在 Form2 对象的 onClose 事件处理过程中加入以下的代码：

```
procedure TForm2.FormClose(Sender: TObject; var Action:
                                          TCloseAction);
begin
  //显示事件名称
  Form1.Memo1.lines.add('现在触发 onClose 事件!!');
end;
```

STEP 8 请在 Form2 对象的 onCloseQuery 事件处理过程中加入以下的代码。这段代码主要是打开一个对话框，让用户决定 CanClose 参数值，换言之，也就是让用户自行决定是否关闭 Form2 对象。

```
procedure TForm2.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  //显示事件名称
  Form1.Memo1.lines.add('现在触发 onCloseQuery 事件!!');

  //由用户来决定 CanClose 的参数值
  if MessageDlg('您确定要关闭这个 Form 吗?', mtConfirmation, [mbOk, mbCancel], 0) =
mrCancel then
    CanClose := False;

  //显示 CanClose 参数值
  if CanClose then
```

```
        Form1.Memo1.lines.add('onCloseQuery 事件的
                                    CanClose 参数值为 True')
      else
        Form1.Memo1.lines.add('onCloseQuery 事件的 CanClose
                                    参数值为 False');
    end;
```

步骤 9 因为我们在 Form2 中直接调用 Form1，因此，我们必须在 Form2 的单元文件(Unit2)中引用 Form1 的单元文件(Unit1)。至于如何引用请参考范例 2-4 的详细说明。

步骤 10 在 Form1 上 TButton 对象的 onClick 事件内容中加入以下的代码。这段代码主要是 Show 出 Form1 对象，并且在 Form2 结束时释放 Form2 对象。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //显示操作方法名称
  Memo1.lines.add('调用 Create 操作方法');
  //动态产生 TForm2 对象
  with TForm2.Create(self) do
  try
    //显示操作方法名称
    Memo1.lines.add('调用 ShowModal 操作方法');
    showModal;
  finally
    //显示操作方法名称
    Memo1.lines.add('调用 Free 操作方法');
    Free;
  end;
end;
```

步骤 11 与步骤 9 一样，因为我们在 Form1 中直接调用 Form2，因此，我们必须在 Form1 的单元文件(Unit1)中引用 Form2 的单元文件(Unit2)。至于如何引用请参考范例 2-4 的说明。

步骤 12 保存工程，并按【F9】功能键运行工程。工程运行后，当您单击“Show Form2”按钮时，系统会动态产生并显示 Form2 对象。当您又单击“Close Form2”按钮时，系统会出现一个询问是否关闭的对话框，只要您选择“OK”按钮，那么 Form2 对象即会释放(Free)，如图 5.16 所示。

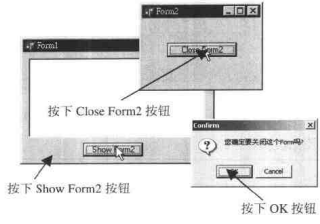


图 5.16　动态产生 TForm 对象



由于在步骤 10 使用 ShowModal 方法，因此在结束 Form2 对象之前，输入焦点是无法回到 Form1 的。由此范例可以得知，一个 TForm 对象从被创建到释放，其共使用了 Create、Close 及 Free 等操作方法，并触发了 onCreate、onCloseQuery、onClose 及 onDestroy 等事件，其运行的顺序结果如图 5.17 所示。



图 5.17　对象从创建到释放的运行顺序图

5.2.2　*TForm* 对象显示与隐藏

Show 操作方法

光是产生 Create 对象还是不够的，Create 操作方法只是将对象产生在内存中(RAM)而已，如果您要将该对象展现出来，您就要使用 Show 操作方法。Show 操作方法是将 TForm 对象的 Visible 属性设为 True，并将 TForm 对象显示在屏幕的最上层。因此如果该 TForm 对象没有被创建 Create，那么使用 Show 会发生错误，如图 5.18 所示。

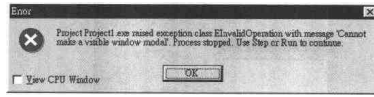


图 5.18　显示(Show)尚未产生的 TForm 对象而产生错误

Show 操作方法的原始类型如下：

```
procedure Show;
```

ShowModal 操作方法

ShowModal 与 Show 是两个类似的操作方法，为的就是要将 TForm 对象加以展现出来。ShowModal 与 Show 最大的不同就是 ShowModal 操作方法所产生的新 TForm 对象必须要结束，才能回到原来的 TForm 对象。ShowModal 操作方法的原始类型如下：

```
function ShowModal: Integer; virtual;
```

注意　如果 ShowModal 操作方法和 ModalResult 属性配合使用，可以很容易获取 ShowModal 操作方法所显示的 TForm 对象是以什么 Modal 结束的。以下将以范例说明如何获取 TForm 对象的 ModalResult 属性值。

5-18

目的：本范例示范如何获取 TForm 对象的 ModalResult 属性值。

1. 创建新的工程。
2. 在预设的 TForm 对象上加入一个 TButton 对象，将此 TButton 对象的 Caption 属性设为“Show Form2”。最后在此 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  myResult : Integer;
begin
  //获取 Form2 的 ModalResult
  myResult := Form2.ShowModal;

  //判断 Form2 的 ModalResult
  if myResult = mrOK then
    showmessage('您按下 OK 按钮')
  else if myResult = mrYES then
    showmessage('您按下 YES 按钮')
  else
```

```
    showmessage('您按其他按钮结束 Form2');
end;
```

③ 参考范例 5-17 的作法，为此工程另外创建一个 TForm 对象，命名为 Form2。并且在此 TForm 对象上加入两个 TButton 对象，其相关属性设置如下：

④ 在 btYES 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  ModalResult := mrYES;
end;
```

⑤ 在 btOK 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm2.Button2Click(Sender: TObject);
begin
  ModalResult := mrOK;
end;
```

⑥ 保存工程，请别忘了在 Form1 的单元文件(unit1)中引用 Form2 的单元文件(unit2)。至于如何引用请参考范例 2-4 的详细说明。

⑦ 保存工程，并按【F9】功能键运行工程。工程运行后，当您结束 Form2 时，系统会根据 Form2 对象的 ModalResult 属性值来判断您是按下哪个按钮来结束 Form2 对象，如图 5.19 所示。

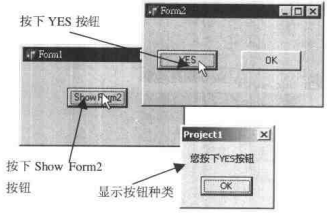


图 5.19　获取 TForm 对象的 ModalResult 属性值

Hide 操作方法

如果只是要将 TForm 对象显示隐藏而已，那您使用 Hide 方法就可以了。Hide 可以和 Show 方法配合使用。Hide 操作方法的原始类型如下：

```
procedure Hide;
```

SendToBack 操作方法与 BringToFront 操作方法

SendToBack 操作方法是将 TForm 对象隐藏到所有相同 Parent 对象的背景，而 BringToFront 操作方法则是将 TForm 对象呈现在所有相同 Parent 对象的前景。SendToBack 操作方法与 BringToFront 操作方法的原始类型如下：

```
Procedure SendToBack;
Procedure BringToFront;
```

范例 5-19

目的：在 Delphi 6 中除了提供很多组件供使用外，更提供一些现成的 Form 对象可以使用，本范例将通过示范“About Form”的作法，来说明 SendToBack 操作方法与 BringToFront 操作方法的不同之处。

步骤 1 创建新的工程。

步骤 2 选择菜单栏中的“File/New”，并选择“Forms”页标签，再点选“About Box”，此时 Delphi 6 自动会产生一个新的 About box Form，如图 5.20 所示。

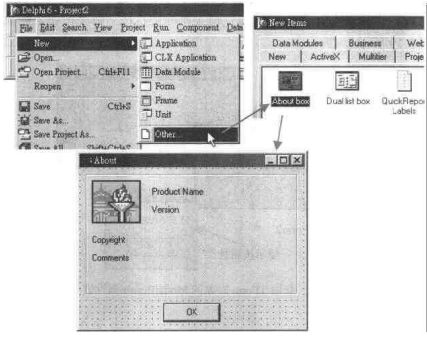


图 5.20 利用 Delphi 6 内置的 Form 模板对象

步骤 3 此时您可以针对 AboutBox Form 上的对象做任意的设置更改，这些操作在后面的章节进行说明，在此先略过。请先在 TAboutBoxForm 中 OKButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TAboutBox.OKButtonClick(Sender: TObject);
```

```
begin
  Close;
end;
```

4 并在 AboutBox 对象上另外加入一个 TButton 对象，Name 属性设为 Button1，Caption 属性分别设为“Send AboutBox To Back”，并在 onClick 事件内容中加入以下的程序：

```
procedure TAboutBox.Button1Click(Sender: TObject);
begin
   SendToBack;
end;
```

5 在 Form1 上加入一个 TButton 对象，Name 属性设为 Button1，Caption 属性设为“关于..”，并在其 onClick 加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AboutBox.show;
end;
```

6 另外在 Form1 对象上加入一个 TButton 对象，Name 属性设为 Button2，Caption 属性分别设为“Bring AboutBox To Front”，并在 onClick 事件内容中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if AboutBox.visible = false then
    showmessage('AboutBox 目前为不可视对象')
  else
    AboutBox.BringToFront;
end;
```

7 在保存工程前可别忘了在 Form1 的单元文件(unit1)的 Uses 子句中加入 AboutBox 的单元文件(unit2)，否则您的 Form1 可不认识 TAboutBox 对象喔！

8 保存工程。

9 按【F9】键运行工程。工程运行后，您只要在 Form1 对象上按 Button1 就可以产生 AboutBox Form。Form1 在 AboutBox Form 结束之前还可以获取焦点，这就是 Show 操作方法与 Showmodal 操作方法最大的不同。另外您也可以利用 Form1 上的“Bring AboutBox to Front”按钮将 AboutBox 对象 Bring to Front，及 AboutBox 对象的“Send Form to Back”按钮将 AboutBox 对象 Send to Back，如图 5.21 所示。



图 5.21 BringToFront 与 SendToBack

Repaint 操作方法

Repaint 操作方法可以重画窗口上对象的图像，Repaint 操作方法原始类型如下：

```
procedure Repaint; override;
```

Refresh 操作方法

Refresh 操作方法可以将 TForm 对象上的控制组件加以重画(repaint)。因为 Refresh 操作方法是调用 Repaint 操作方法来重画 TForm 对象上的对象，因此 Refresh 与 Repaint 两者可以替换使用。Refresh 操作方法原始类型如下：

```
procedure Refresh;
```

Print 操作方法

Print 可以将 TForm 对象以 BitMap(BMP)格式打印，Print 操作方法会调用 GetFormImage method 来完成工作。Print 操作方法原始类型如下：

```
procedure Print; virtual;
```

范例 5-20

目的：本范例在示范如何将使用 TForm 对象的 Print 属性。

(1) 创建新的工程。

(2) 在 TForm 对象上加入一个 TButton 对象，Caption 设为“打印窗体”，在 onClick 事件内容中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Print;
end;
```

(3) 保存工程。

(4) 按【F9】键运行工程，只要您按下 TButton 对象，系统便会自动的将 TForm 对象(包

含所有对象)加以打印。不过经过我的测试，发觉到打印的质量可能不是很理想!

5.2.3　*TForm* 对象产生方式

在第 2 章中我们曾经说明工程(Project)中有一个 TApplication 对象，它是负责管理此应用程序的相关信息，以下用一个范例来说明 TApplication 如何管理自己所属的对象。此范例上要延续自范例 5-17，相同的设置就不再赘述。

范例 5-21

目的：本范例主要测试在第二个 TForm 对象调用 Create 前后对整个应用程序的影响。

步骤 1　本范例延续自范例 5-17，其他步骤请自行参考该范例。

步骤 2　在 Form1 对象上加入一个 TButton 对象，并将其 Name 属性设为 Button2、Caption 属性设为“显示 TApplication 的对象”，最后请在其 onClick 事件处理过程中加入以下的代码。这段代码主要是显示目前整个 Application 中所有 TForm 对象的 Name 属性。

```
procedure TForm1.Button2Click(Sender: TObject);
var
   I:Integer;
   myString:String;
begin
  with Application do
  For I:=0 to ComponentCount-1 do
    if Components[I] is TForm then
      MyString:= MyString + TForm(Components[I]).Name+ #10;
  ShowMessage(MyString);
end;
```

步骤 3　在 Form2 对象上加入一个 TButton 对象，并将其 Name 属性设为 Button2、Caption 属性设为“显示 TApplication 的对象”，最后请在其 onClick 事件处理过程中加入与 Form1 中 Button2 对象的 onClick 事件相同的代码。为什么要写两份完全相同的代码呢?其实主要的目的是要测试在 Form2 对象 Create 前后对整个应用程序有何不同。

步骤 4　保存工程。

步骤 5　按【F9】功能键运行程序，并且测试产生 Form2 对象对 Application 的影响。

注意　此时您可能会很惊讶，为什么 Form2 对象在 Create 前后似乎对整个应用程序没什么改变？也就是说，在产生 Form2 对象之后，整个 Applicationorm 所包含的组件(Components)似乎没有改变。这其中的原因在于 Form2 对象产生的方式，因为我们使用 Self 作为 TForm2.Create 操作方法的参数，而 self 在这里代表 Form1 对象(关于 Self 的用法请参考第 3.7 节)。因此我们只要将 Self 更改为 Application 即可，更改后的运行结果如图 5.22 所示。

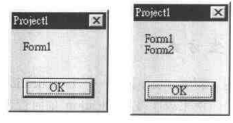


图 5.22 Create 操作方法对 Application 的影响

5.2.4 TForm 对象焦点(Focus)设置

SetFocus 操作方法与 CanFocus 操作方法

SetFocus 操作方法可以让 TForm 对象获取输入的焦点。但决定 SetFocus 操作方法是否有效是由 CanFocus 操作方法的返回值来决定，当 CanFocus 操作方法返回值为 True 时，则 SetFocus 操作方法有效，反之，则无效。SetFocus 操作方法与 CanFocus 操作方法的原始类型如下：

```
procedure SetFocus;
function CanFocus: Boolean;
```

Focused 操作方法

Focused 操作方法可以让用户获取 TForm 对象是否已获取输入焦点，Focused 操作方法原始类型如下：

```
function Focused: Boolean; dynamic;
```

5.2.5 对象控制的操作方法

RemoveComponent 操作方法

从 TForm 的 Component List 中删除一个指定的组件，Remove Componect 操作方法原始类型如下：

```
procedure RemoveComponent(AComponent: TComponent);
```

RemoveControl 操作方法

从 TForm 的 Control List 中删除一个指定的组件，RemoveControl 操作方法原始类型如下：

```
procedure RemoveControl(AControl: TControl);
```

范例 5-22

目的：本范例示范如何在运行时期增加、删除对象。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入 TEdit、TButton 及 TGroup 等三个对象，如表 5.15 所示。

表 5.15

对象种类	Name　属性	Caption　属性
TEdit	Edit1	
TButton	Button1	MoveControl
Tgroup	Group1	

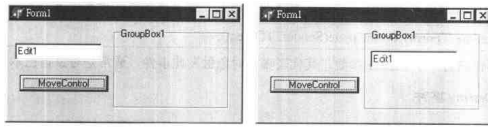
步骤 3 在 TButton 对象的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  RemoveControl(Edit1);
  GroupBox1.InsertControl(Edit1);

  //上面的代码，可以用以下的替代
  //Edit1.Parent := GroupBox1;
end;
```

步骤 4 保存工程。

步骤 5 按【F9】键运行工程，工程运行后，您可以按下 Button1，将 Edit1 对象由 Form1 移到 Group1 对象上，如图 5.23 所示。



移动前　　移动后

图 5.23　动态移动对象

5.3　TForm 对象的事件处理

在前面的章节中，笔者已经说明 TForm 对象的部分事件，也实际上运用它们。在这个小节中，笔者将对 TForm 对象常用的事件加以归类说明。

5.3.1 鼠标相关事件

TForm 对象与鼠标有关的事件包括：OnClick、OnDblClick、OnMouseDown、OnMouseMove、OnMouseUp、OnMouseWheel、OnMouseWheelDown 与 OnMouseWheelUp 等事件。这些鼠标事件几乎大部分为可视对象的共同事件，相关说明请参考 4.2 小节的详细说明。

5.3.2 键盘相关事件

TForm 对象与键盘有关的事件包括：OnKeyDown、OnKeyPress 与 OnKeyUp 等事件。这些键盘事件大部分为可视对象的共同事件，相关说明请参考 4.3 小节的详细说明。

5.3.3 拖动相关事件

TForm 对象与拖动有关的事件包括：OnDockDrop、OnDockOver、OnStartDock、OnEndDock、OnUnDock、OnDragDrop 及 OnDragOver 等事件。这些拖动事件大部分也是可视对象的共同事件，相关说明请参考第 3.5 节的详细说明。

5.3.4 创建与释放相关事件

TForm 对象从创建到释放都有相关的事件，以供控制 TForm 对象运行时期每一个细微的变化。通过如此的控制，您就可以快速地写出很专业的程序。TForm 对象创建与释放的相关事件包括：OnCreate、OnDestroy、OnActivate、OnDeactivate、OnCloseQuery、OnClose、OnHide、OnShow 及 OnPaint 等事件，分述如下：

- *OnCreate* 事件

当 TForm 对象被创建时 onCreate 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormCreate(Sender: TObject);
```

此事件只有一个 Sender 参数，其代表哪个对象触发此事件，通常是对象自己本身。

- *OnDestroy* 事件

当 TForm 对象被释放时 onDestroy 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormDestroy(Sender: TObject);
```

此事件只有一个 Sender 参数，其代表哪个对象触发此事件，通常是对象自己本身。

- *OnActivate* 事件

当 TForm 对象获取焦点时 onActivate 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormActivate(Sender: TObject);
```

此事件只有一个 Sender 参数，其代表哪个对象触发此事件，通常是对象自己本身。

OnDeactivate 事件

当 TForm 对象失去焦点时 onDeactivate 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormDeactivate(Sender: TObject);
```

此事件只有一个 Sender 参数，其代表哪个对象触发此事件，通常是对象自己本身。

OnCloseQuery 事件

OnCloseQuery 事件是一个询问性质的事件，当过程调用 TForm 对象的 Close 操作方法时 onCloseQuery 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormCloseQuery(Sender: TObject;
var CanClose: Boolean);
```

此事件有 Sender 与 CanClose 两个参数，前者代表哪个对象触发此事件，通常是对象自己本身。后者代表是否关闭此对象，假如 CanClose 参数被设为 False 则表示不关闭此对象，反之则关闭。系统默认值为 True。

OnClose 事件

当过程调用 TForm 对象的 Close 操作方法，而且 onCloseQuery 事件的 CanClose 参数为 True 时，onClose 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormClose(Sender: TObject;
var Action: TCloseAction);
```

此事件有 Sender 与 Action 两个参数，前者代表哪个对象触发此事件，通常是对象自己本身。后者代表关闭此对象的方法，其可能的方式有 caNone、caHide、caFree 及 caMinimize 等。caNone 代表不关闭此对象，也就是说，什么事也没发生；caHide 代表只是把此对象隐藏而已，并没有真正的关闭；caFree 代表真正将此对象从内存中释放；caMinimize 代表只是将此对象最小化而已。在一般的应用程序中系统默认值为 caHide，而在 MDI(Multiple Document Interface，多重文件接口)的应用程序中，其系统默认值为 caMinimize。

OnHide 事件

当过程调用 TForm 对象的 Hide 操作方法时 onHide 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormHide(Sender: TObject);
```

此事件只有 Sender 一个参数，代表哪个对象触发此事件，通常是对象自己本身。

OnShow 事件

当过程调用 TForm 对象的 Show 操作方法时 onShow 事件即被触发，此事件的原始声明

如下：

```
procedure TForm1.FormShow(Sender: TObject);
```

此事件只有 Sender 一个参数，代表哪个对象触发此事件，通常是对象自己本身。

OnPaint 事件

当 TForm 对象被创建时或其他对象经过时 onPaint 事件即被触发，此事件的原始声明如下：

```
procedure TForm1.FormPaint(Sender: TObject);
```

此事件只有 Sender 一个参数，代表哪个对象触发此事件，通常是对象自己本身。

范例 5-23

目的：为了让大家能充分了解 TForm 对象从创建到释放相关的事件，笔者特别以此范例来说明每一个事件间的关系。

步骤 1 创建新的工程，其预设的 TForm 对象名称为 Form1。

步骤 2 在 Form1 对象上加入一个 TMemo 对象，并清除 TMemo 对象的 Lines 属性值。为此工程新增另一个 TForm 对象，其预设的名称为 Form2。

步骤 3 在 Form1 对象上加入三个 TButton，其 Name 属性分别为 ButtonHide、ButtonShow 及 ButtonCreate。

步骤 4 将 ButtonHide 的 Caption 属性改为“Hide Form2”，并在其 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonHideClick(Sender: TObject);
begin
  //将 Form2 对象加隐藏
  Form2.Hide;
end;
```

步骤 5 将 ButtonShow 的 Caption 属性改为“Show Form2”，并在其 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonShowClick(Sender: TObject);
begin
  //显示 Form2 对象
  Form2.Show;
end;
```

步骤 6 将 ButtonCreate 的 Caption 属性改为“Create Form2”，并在其 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonCreateClick(Sender: TObject);
var
```

```
  tmpForm : TForm2;
begin
  //以动态的方式创建另一个 TForm2 对象
  tmpForm := TForm2.Create(Application);
  try
    tmpForm.ShowModal;
  finally
    tmpForm.Free;
  end;
end;
```

步骤7 在Form1与Form2的onActivate事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onActivate Event');
```

步骤8 在Form1与Form2的onCreate事件处理过程加入以下的代码，在这段代码中，笔者特别判断其Action方式为何，并显示在Form1的Tmemo对象中：

```
var
  ActionStr : String;
begin
  case Action of
   caNone : ActionStr := 'caNone';
   caHide : ActionStr := 'caHide';
   caFree : ActionStr := 'caFree';
   caMinimize : ActionStr := 'caMinimize';
  end;

  Form1.Memo1.Lines.Add((Sender as TForm).Name +
' onClose Event, Action=' + ActionStr);
```

步骤9 在Form1与Form2的onCloseQuery事件处理过程加入以下的代码，在这个事件处理过程中，笔者加入一段询问的程序，可以让用户在程序的运行过程中自行决定是否关闭该TForm对象。

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onCloseQuery Event');
if MessageDlg('您要关闭此 TForm 对象吗?',mtConfirmation,
              [mbYES,mbNo],0) = mrNo then
  CanClose := False;
```

步骤10 在Form1与Form2的onCreate事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onCreate Event');
```

步骤11 在Form1与Form2的onDeactivate事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onDeactiveate Event');
```

step 12 在 Form1 与 Form2 的 onDestroy 事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onDestroy Event');
```

step 13 在 Form1 与 Form2 的 onHide 事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onHide Event');
```

step 14 在 Form1 与 Form2 的 onPaint 事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onPaint Event');
```

step 15 在 Form1 与 Form2 的 onShow 事件处理过程加入以下的代码：

```
Form1.Memo1.Lines.Add((Sender as TForm).Name + ' onShow Event');
```

step 16 保存工程并按下【F9】功能键运行工程。由于在 Form1 对象中直接调用 Form2 对象，而且在 Form2 对象中也有调用 Form1 对象，所以系统会出现如图 5.24 所示的询问对话框，并且会出现两次（见图 5.25）。请记得两次都单击“Yes”钮。如果您重新查看代码，应该会发现系统分别在 Unit1 与 Unit2 中自动帮您写入下列的“uses Unit1;”与“uses Unit2;”的程序。

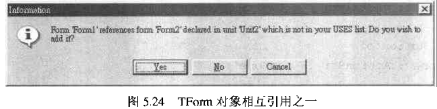


图 5.24 TForm 对象相互引用之一

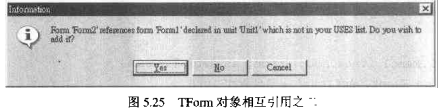


图 5.25 TForm 对象相互引用之二

step 17 重新再按下【F9】功能键运行工程。结果如图 5.26 所示，一般来说创建一个 TForm 对象应该会有 onCreate、onShow、onActivate 及 onPaint 事件。



图 5.26 创建 TForm 对象所触发的相关事件



此时您可能会很惊讶，为什么在画面上看不到 Form2 对象？可是在 TMemo 对象中竟然出现 Form2 对象被创建的事件呢？事实上，Form2 对象已经被加载内存(RAM)，就如同笔者在第 2.2.2 节所述叙的一样。因为 Form2 对象被默认为 Auto-Create form，也就是说当程序被运行时，这个对象就必须被加载内存。相关的设置方法请参考第 2.2.2 节。

在步骤 7，程序可以通过动态产生方式产生 TForm2 对象，通过动态的创建，我们可以了解创建一个 TForm 对象会触发 onCreate、onShow、onActivate 及 onPaint 等事件。相对的，释放一个 TForm 对象则会触发 onCIsoeQuery、onClose、onDeactivcatc、onHide 及 onDestroy 等事件，如图 5.27 所示。

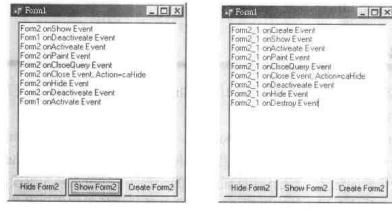


图 5.27　创建 TForm 对象所触发的相关事件

从上图中，您可以发现 Form2 与动态产生的 Form2_1，在整个创建与释放的过程中所触发的事件有些稍微的不同。那是因为 Form2 在程序运行时就加载内存，而 Form2_1 则是动态产生的。另外一个很大的不同是，当你关闭 Form2 与 Form2_1 时，前者只是隐藏而已；而后者则是通过步骤 7 的 Free 操作方法将其真正的在记忆中释放。

5.3.5　其他相关事件

除了上述的事件外，TForm 对象还有 OnShortCut、OnCanResize、OnConstrainedResize、OnResize 及 OnContextPopup 等常用的事件，分述如下：

- *OnShortCut* 事件

在前面的小节笔者介绍过与键盘相关的事件处理，事实上，当用户按下某个按键时，系统会先触发 onShortCut 事件，再触发 onKeyDown 事件。onShortCut 事件可以让您事先处理一些特定按键所触发的事件，例如 F1 到 F12 的功能按键、Pause 特殊键、Enter、Tab、Caps lock 等按键。onShortCut 事件的原始声明如下：

```
procedure TForm1.FormShortCut(var Msg: TWMKey;
var Handled: Boolean);
```

此事件只有 Msg 与 Handled 两个参数，代表用户所按下的按键，后者代表此事件处理过程是否处理这个事件。Msg 参数是一个 TWMKey 类型的变量，其原始声明如下：

```
type
  TWMKey = packedrecord
    Msg: Cardinal;
    CharCode: Word;
    Unused: Word;
    KeyData: Longint;
    Result: Longint;
  end;
```

笔者将为大家说明 TWMKey 类型中的 Msg 与 CharCode。其中 Msg 代表 Windows 的信息代码(message code)；CharCode 代表用户所按下的按钮的虚拟代码(virtual key code)，通过 CharCode 我们可以得知用户所按下的按钮是什么。

范例 5-24

目的：本范例将示范如何通过 TForm 对象的 onShortCut 事件获取用户所按下的各种按键。

STEP 1 创建新的工程。

STEP 2 在 Form1 对象上加入一个 TMemo 对象，并清除 Tmemo 对象的 Lines 属性值。

STEP 3 创建一个 TransferCharCode 子程序，以便将 onShortCut 事件的 CharCode 参数值转换为所代表的意义。要创建用户自定义的子程序必须分为两个阶段：第一阶段是声明，第二阶段是程序的实现部分。由于系统所提供的虚拟代码很多，基于篇幅的考虑，以下仅列出部分的程序片段，详细的代码请参考光盘范例：

```
  public
    { Public declarations }
    //声明一个用户自定义的子程序
    function TransferCharCode(CharCode : Word):String;
{略}

//子程序实现部分
function TForm1.TransferCharCode(CharCode : Word):String;
begin
  case CharCode of
    VK_BACK : Result := 'Backspace key';
    VK_TAB : Result := 'Tab key';
    VK_RETURN : Result := 'Enter key';
    VK_SHIFT : Result := 'Shift key';
    VK_CONTROL : Result := 'Ctrl key';
    VK_MENU : Result := 'Alt key';
    VK_PAUSE : Result := 'Pause key';
    VK_CAPITAL : Result := 'Caps Lock key';
```

```
      VK_ESCAPE : Result :=  'Esc key';
      VK_SPACE : Result :=  'Space bar';
      VK_PRIOR : Result :=  'Page Up key';
      VK_NEXT : Result :=  'Page Down key';
      VK_END : Result :=  'End key';
      VK_HOME : Result :=  'Home key';
      VK_LEFT : Result :=  'Left Arrow key';
      VK_UP : Result :=  'Up Arrow key';
      VK_RIGHT : Result :=  'Right Arrow key';
      VK_DOWN : Result :=  'Down Arrow key';
      {略}
      else  Result := '其他键=' + IntToStr(CharCode);
    end;
  end;
```

④ 在 Form1 对象的 onShortCut 事件处理过程中加入以下的代码：

```
procedure TForm1.FormShortCut(var Msg: TWMKey; var Handled: Boolean);
var
  tmp : TWMKey;
begin
  tmp := Msg;
  Memo1.Lines.Append('onShortCut=' +
TransferCharCode(tmp.CharCode));
  Handled := true;
end;
```

⑤ 按【F9】功能键运行工程。其结果如图 5.28 所示。通过 onShortCut 事件处理过程，我们可以很轻易地获取用户所按下的任何特殊按键，包括 Page Up、Page Down 及 Delete 等。



图 5.28　onShortCut 事件处理范例

范例 5-25

目的：本范例将示范如何通过 TForm 对象的 onShortCut 事件让程序在运行时期允许用户以【Enter】键改变光标焦点。当然，您只要稍微改写本范例的代码，此方法同样适合于 onKeyPress 事件。

STEP 1 创建新的工程。

STEP 2 在 Form1 对象上分别加入 TEdit、TComboBox、TCheckBox、TRadioButton、TButton 及 TMaskEdit 等对象。加入这么多种类似对象，为的只是要证明此范例所示范的方法适用于任何对象。

STEP 3 在 Form1 对象的 onShortCut 事件处理过程中加入以下的代码，这些程序主要分为四个部分。第一、判断用户所按下的按键是否为【Enter】键；第二、获取目前已获取焦点的对象；第三、判断此组件是否为最后一个，如果是，则跳回第一个对象；第四、为下一个对象设置焦点。

```
procedure TForm1.FormShortCut(var Msg: TWMKey; var Handled: Boolean);
var
  i : integer;
begin
  //判断用户所按下的是否为 Enter 按键
  if Msg.CharCode = VK_RETURN then begin
    //获取目前已获取焦点的对象
    i := ActiveControl.ComponentIndex+1;
    //判断组件是否为最后一个，如果是则跳回第一个
    if i = ComponentCount then i := 0;
    //为下一个对象设置焦点
    TWinControl(Controls[i]).SetFocus;
  end;
  //代表此事件处理过程会处理这个 ShortCut 事件
  Handled := true;
end;
```

STEP 4 保存并按【F9】功能键运行工程。工程运行后，您可以通过【Enter】键来循序改变光标焦点，如图 5.29 所示。

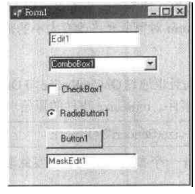


图 5.29　通过 onShortCut 事件处理改变光标焦点范例

OnContextPopup 事件

当用户在 TForm 对象上按下右键以显示 PopupMenu 前，系统会触发 onContentPopup 事件，OnContentPopup 事件的原始声明如下：

```
Procedure TForm1.FormContextPopup(Sender: TObject;
MousePos: TPoint; var Handled: Boolean);
```

此事件处理过程包含 Sender、MousePos 与 Handled 三个参数。Sender 参数代表触发此事件的对象，通常是对象本身；MousePos 参数代表目前鼠标的坐标位置；Handled 参数则代表是否处理此事件。

OnCanResize 事件

当 TForm 对象被改变大小时，其依序可能触发 onCanResize、OnConstrainedResize 及 onResize 等事件。onCanResize 事件的原始声明如下：

```
procedure TForm1.FormCanResize(Sender: TObject; var NewWidth,
  NewHeight: Integer; var Resize: Boolean);
```

此事件处理过程包含 Sender、NewWidth、NewHeight 与 Resize 等四个参数。Sender 参数代表触发此事件的对象，通常是对象本身；NewWidth 与 NewHeight 等参数代表 TForm 对象即将被改变的宽度和高度；Resize 参数则代表是否中断改变过程，当 Resize 参数值为 False 时，则系统不会再触发 OnConstrainedResize 及 onResize 等事件。因此，您可以通过此事件来决定 TForm 对象在运行时期所能改变的大小范围。

OnConstrainedResize 事件

OnConstrainedResize 与 OnCanResize 事件具有相同的功能，前者允许您通过参数值的设置，来决定 TForm 对象在运行时期所能改变的大小范围。OnConstrainedResize 事件的原始声明如下：

```
procedure TForm1.FormConstrainedResize(Sender: TObject;
 var MinWidth, MinHeight, MaxWidth, MaxHeight: Integer);
```

此事件处理过程包含 Sender、MinWidth、MinHeight、MaxWidth 及 MaxHeight 等参数。

Sender 参数代表触发此事件的对象，通常是对象本身；而 MinWidth、MinHeight、MaxWidth 及 MaxHeight 则分别代表此 TForm 对象所允许改变大小的范围。

OnResize 事件

当用户欲改变TForm对象而且也通过 OnCanResize 与 OnConstrainedResize 等事件的限制条件后，系统会触发 onResize 事件。onResize 事件的原始声明如下：

```
procedure  TForm1.FormResize(Sender: TObject);
```

此事件处理过程只有 Sender 一个参数，其代表触发此事件的对象，通常是对象本身。

范例 5-26

目的：本范例将示范如何在运行时期限制 TForm 对象所能改变的大小范围。

步骤 1 创建新的工程。

步骤 2 在 Form1 对象加入一个 TMemo 对象。

步骤 3 在 Form1 对象的 onCanResize 事件处理过程中加入以下的代码：

```
Memo1.Lines.Append('NewWidth=' + IntToStr(NewWidth) +
', NewHeight=' + IntToStr(NewHeight));
```

步骤 4 在 Form1 对象的 onConstrainedResize 事件处理过程中加入以下的代码：

```
MinWidth := 100;
MinHeight := 100;
MaxWidth := 500;
MaxHeight := 500;
```

步骤 5 在 Form1 对象的 onResize 事件处理过程中加入以下的代码：

```
Memo1.Lines.Append('Resized');
```

步骤 6 按下【F9】功能键运行工程，工程运行后，在预设的范围中您可以任意的改变 Form1 的大小，如图 5.30 所示。

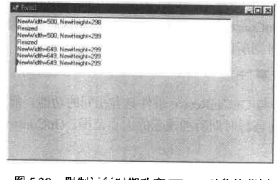


图 5.30 限制运行时期改变 TForm 对象的范例

第6章

Delphi 6 按钮控制对象

本章目标

在前面的几个章节中，笔者曾多次使用 TButton 对象。事实上 Delphi 6 除了 TButton 对象外，另外提供了多个按钮对象。在本章中我们主要说明有关 Delphi 6 所提供的按钮组件，按钮对象算是窗口程序设计最常被使用的输入控件，这些按钮对象包括 TButton、TBitButton、TToolBar、TCoolBar、TupDown、TSpinEdit、TSpeedButton 及 TSpinButton 等组件。

本章主题

6.1 Tbutton 对象

6.2 TbitButton 对象

6.3 TtoolBar 对象

6.4 TcoolBar 对象

6.5 TupDown 对象与 TspinEdit 对象

6.6 TspeedButton 对象

6.7 TspinButton 对象

6.1 TButton 对象

TButton 对象是最常被使用的按钮组件，位于对象面板上的 Standard 页标签中。在前面每一个章节中，我们几乎每一个范例都有使用到它，相信您对它的 Name、Caption、Left、Top、Width 及 Height 等属性及 onClick 的事件处理过程(Event)一定不会陌生才是。以上这些您已经知道的就不再赘述了。本小节将为大家说明 Default(是否为默认按钮)、Cancel(是否为默认的取消按钮)及 ModalResult(结束模式)等三个最常被使用的属性。

Default 属性

Default 属性是决定此 TButton 对象是否为 TForm 对象的默认按钮。当用户在 TForm 对象上按下【Enter】键的同时，会引发这个默认按钮的 onClick 事件。

Cancel 属性

Cancel 属性与 Default 属性类似，惟一不同的是所对应的对象，Default 属性是针对【Enter】键，而 Cancel 属性是针对【Esc】键。若此 TButton 对象是 TForm 对象的默认取消按钮，当用户在 TForm 对象上按下【Esc】键的同时，会引发这个默认按钮的 onClick 事件。



范例 6-1

目的： 本范例目的在示范如何将 TButton 设为 TForm 对象的默认按钮(【Enter】键)及默认取消按钮(【Esc】键)。

1. 创建新的工程。
2. 在 TForm 对象中加入两个 TButton 对象，其属性设置如表 6.1 所示。

表 6.1

对象(name 属性)	属性名称	属性设置值
TButton(Button1)	Default	True
TButton(Button2)	Cancel	True

3. 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   showMessage('您按下 Enter 键来结束 TForm 对象');
   Close;
end;
```

4. 在 Button2 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   showMessage('您按下 ESC 键来结束 TForm 对象');
```

```
    Close;
end;
```

步骤 5 保存工程。

步骤 6 按【F9】运行工程。工程运行后，只要您在 TForm 对象上按下【Enter】键就会触发 Button1 的 onClick 事件；若您按下【Esc】键将会触发 Button2 的 onClick 事件。

注意 运行这个范例时，您必须特别注意焦点(Focus)的问题。假如焦点在 Button2 时而且您也按下【Enter】键时，系统会触发 Button2 的 onClick 事件，而非是 Button1 的 onClick 事件了。

ModalResult 属性

当您使用 ShowModal 的方法打开一个 TForm 对象，如果您要关闭这个 TForm 对象时，您可以利用 Close 来关闭它。除此之外您有更好的方法，那就是用 TButton 的 ModalResult 属性来完成，用这个方法您完全不需要写任何程序，而且可以完全掌握用户到底是用哪一个按键来结束这个 TForm 对象的。ModalResult 属性共有 10 个属性值，如表 6.2 所示。

表 6.2

ModalResult 属性值
mrNone
mrOk
mrCancel
mrAbort
mrRetry
mrIgnore
mrYes
mrNo
mrAll
mrNoToAll
mrYesToAll

范例 6-2

目的：本范例利用 Form1 来调用 Form2，再利用 TButton 对象的 ModalResult 属性来结束 Form2 对象、结束时并显示 Form2 的结束模式。

步骤 1 创建新的工程。

步骤 2 除了工程默认的 TForm 对象 Form1 外，再利用菜单栏中“File/New/Form”加入一个新的 TForm 对象 Form2。

步骤 3 在 Form1 对象上加入一个 TButton 对象，将 Caption 属性设为“Call Form2”，且在

onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    Form2.ShowModal;
  finally
    case Form2.ModalResult of
        mrOk     :showMessage('您按下 OK 按钮结束 Form2');
        mrCancel:showMessage('您按下 Cancel 按钮结束 Form2');
        mrAbort  :showMessage('您按下 Abort 按钮结束 Form2');
    else
        showMessage('您按下其他键结束 Form2');
    end;
  end;
end;
```

步骤 4 在 Form2 对象上加入三个 TButton 对象，其属性设置如表 6.3 所示。

表 6.3

对象(name 属性)	属性名称	属性设置值
TButton(Button1)	ModalResult	mrOK
	Caption	OK
TButton(Button2)	ModalResult	mrCancel
	Caption	Cancel
TButton(Button3)	ModalResult	mrAbort
	Caption	Abort

步骤 5 保存工程。

步骤 6 按下【F9】功能键运行工程，工程运行后，您可以调用 Form2 对象，并正确地掌握用户是按下哪个按钮(TButton)结束 Form2 对象。

注意 从本章开始笔者会陆续介绍 Delphi 6 所提供的输入对象，这些输入对象大部分都会有 onEnter 与 onExit 等事件。当指定对象取得焦点时会触发 onEnter 事件；当指定对象失去焦点时会触发 onExit 事件。

6.2 TBitButton 对象

TBitButton 对象是一种比较漂亮的按钮组件，位于组件面板上的 Additional 页标签中。TBitButton 对象与 TButton 对象有很多的相同属性，其最大的不同就是 TBitButton 对象可以显示小图标(Icon)。因此本小节仅说明 TBitButton 对象的特有属性，包括 Kind、ModalResult、Glyph、Margin、Spacing 及 Layout 等六个最常被使用的属性。

Kind 与 ModalResult 属性

在前面我们曾经说明 TBitButton 对象与 TButton 对象最大的不同在于 TBitButton 对象拥有比较活泼的用户接口。Delphi 6 提供了 11 种常用的默认图标，每一种默认图标都对应一种 ModalResult，就如同第 6.1 节所述，ModalResult 属性必须配合 ShowModal 操作方法才会有效，详细说明参考第 6.1 节。表 6.4 将说明 Delphi 6 所默认的图标 Kind 属性及 ModalResult 属性。

表 6.4

Kind 属性值	对应图标	ModalResult 属性值
bkAbort	Abort	mrAbort
bkAll	All	mrAll
bkCancel	Cancel	mrCancel
bkClose	Close	mrClose
bkHelp	Help	mrHelp
bkIgnore	Ignore	mrIgnore
bkNO	No	mrNo
bkOK	OK	mrOk
bkRetry	Retry	mrRetry
bkYES	Yes	mrYes
bkCustom	自行设置	自行设置

Glyph 属性

如果您选择了 Delphi 6 默认的图标(Kind 属性)，那就会拥有一个默认的小图标。如果您想要为自己所开发的应用程序加入自己风格的小图标，那您就必须通过 Glyph 属性来设置自己的图标。以下将示范如何为 TBitButton 加入自己的小图标。

1. 在 TForm 对象上加入一个 TBitButton 对象，并点选其 Glyph 属性值字段上的按钮，此时会打开“Picture Editor”对话框，如图 6.1 所示。

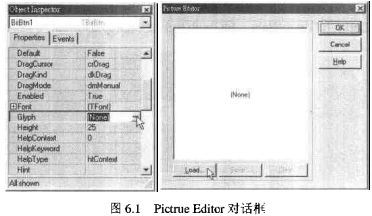


图 6.1　Pictrue Editor 对话框

(2) 在“Picture Editor”对话框中点选“Load...”按钮，打开“Load Picture”对话框，此时选择您所指定的文件，并单击“打开”按钮回到“Picture Editor”对话框。以（\Borland\Delphi6\Demos\ Doc\Graphex\brush.bmp）为例，如图 6.2 所示。

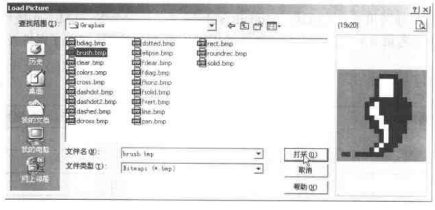


图 6.2 Load Picture 对话框

(3) 在“Picture Editor”对话框单击“OK”按钮即可，此时 TBitButton 对象的小图标便自动的更新为指定的图案了。

Margin 与 *Spacing* 属性

Margin 属性主要是决定 TBitButton 对象中图标(BitMap)与按钮边界的距离，其距离以像素(pixel)为单位。而 Spacing 属性主要是决定 TBitButton 对象中图标(BitMap)与文字说明的距离，其距离以像素(pixel)为单位，如图 6.3 所示。



图 6.3 Margin 及 Spacing 属性

Layout 属性

Layout 属性主要是决定 TBitButton 对象中图标(BitMap)与文字说明的排列方式，Layout 属性共有 blGlyphLeft、blGlyphRight、blGlyphTop 及 blGlyphBottom 等四种，如图 6.4 所示。



图 6.4 Layout 各种属性值

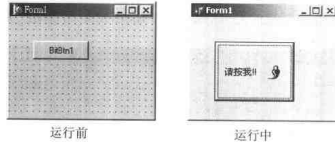
范例 6-3

目的：前面说明了 TBitButton 对象各项常用属性设计时期的设置方法。本范例主要目的是说明如何在运行时期改变 Kind、Glyph、Margin、Spacing 及 Layout 等各项属性。

1. 创建新的工程。
2. 在 TForm 对象上加入一个 TBitButton 对象。
3. 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var Path:String;
begin
 //设置图片文件路径
 Path:='c:\Program files\Borland\Delphi6\Demos\Doc\Graphex\
        brush.bmp';
 with BitBtn1 do
     begin
       Layout:=blGlyphRight;              {设置 Glyph 属性}
       Width:=105;                        {宽度}
       Height:=81;                        {高度}
       Caption:='请按我!!';               {设置文字说明}
       Glyph.LoadFromFile(Path);          {读入图片}
       Margin:=15;                        {设置间隔距离}
       Spacing:=15;
     end;
end;
```

4. 保存工程。
5. 按下【F9】功能键运行工程。工程运行后，所显示出的 TBitButton 对象如图 6.5 所示。



运行前　　运行中

图 6.5　运行时期设置 TBitButton 各种属性值

6.3　TToolBar 对象

TToolBar 对象提供一种非常便利的方法让您创建及管理按钮对象，位于对象面板上的

Win32 页标签中。它同时也提供您快速制作类似于 Microsoft Office97/2000 非常酷的按钮对象。一般来说使用 TToolBar 对象当作工具栏有几点好处：

- 保证工具栏(TToolBar 对象)上所有的按钮对象有相同高度与宽度。
- 当工具栏的按钮对象超过窗口宽度(TForm 对象)会自动折到上一行。
- 提供一种特殊的分隔对象，可以将工具栏上的按钮对象加以分为不同组。
- 配合 TImageList 对象可以为工具栏上的按钮对象加入各种不同的图片。

虽然用 TToolBar 对象来当工具栏有以上的好处，但其终究还是由按钮对象 TToolButton 所组成的，因此我们在前面两小节所讨论的 TButton 及 TBitButton 对象的大部分属性及事件都适用。所以接着就 TToolBar 及 TToolButton 对象的特有属性加以说明(参见图 6.6)：

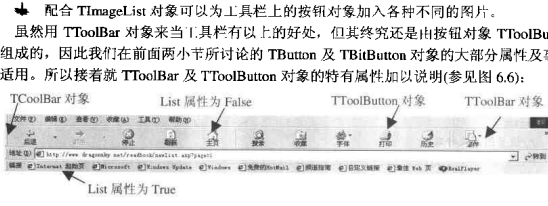


图 6.6 TToolBar 各种属性值

6.3.1 TToolBar 对象属性

Align 属性

Align 这个属性相信您一定不会陌生。Align 属性是提供 TToolBar 对象各种不同排列方式的属性，其属性值共有 alNone、alTop、alBottom、alLeft、alRight、alClient 等。

AutoSize 属性

在前面曾经说过 TToolBar 对象是由数个 TToolButton 对象所组合而成的。AutoSize 属性可以决定 TToolBar 对象的大小是否会随着 TToolButton 而自动改变。假如 AutoSize 属性值为 True 表示 TToolBar 对象的大小会随着 TToolButton 而自动改变；反之，当 AutoSize 属性值为 False 时，则表示 TToolBar 对象的大小不会随着 TToolButton 而自动改变，如图 6.7 所示。



图 6.7 TToolBar 对象 AutoSize 属性值

BorderWidth 属性

BorderWidth 属性是决定 TToolBbutton 对象与 TToolBar 对象之间的距离。您可以直接输

入数字在 Border 属性栏中，或是在设计时期指定 Border 属性值，如图 6.8 所示。



图 6.8　TToolBar 对象 BorderWidth 属性值

EdgeBorder 属性

EdgeBorder 属性是用来决定 TToolBar 对象所出现的边框，其属性值有 ebLeft(左边框)、ebRight(右边框)、ebTop(上边框)及 ebBottom(下边框)等。

EdgeInner 与 EdgeOuter 属性

EdgeInner 与 EdgeOuter 分别用来设置 TToolBar 对象的内、外边框类型。共分为 esNone(无边框)、esRaised(凸起)及 esLowered(凹下)等。这两个属性必须配合 EdgeBorder 属性使用，如图 6.9 所示。

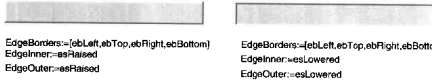


图 6.9　TToolBar 对象 EdgeBorder、EdgeInner 及 EdgeOuter 属性值

List 属性

List 属性是用来决定 TToolButton 对象在 TToolBar 对象上的显示方式，当 List 属性值为 True 表示 TToolButton 对象的图片与文字以水平(左右)方式显示；反之，当 List 属性为 False，则表示 TToolButton 对象的图片与文字以垂直(上下)方式显示，如图 6.10 所示。



图 6.10　TToolBar 对象 List 属性值

Indent 属性

Indent 属性是用来决定 TToolButton 对象与 TToolBar 对象之间的距离，如图 6.11 所示。

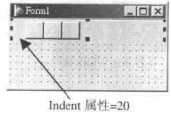


图 6.11　TToolBar 对象 Indent 属性值

Transparent 属性

Transparent 属性是用来决定 TToolBar 对象是否为透明对象。当 Transparent 属性值为 True，表示 TToolBar 对象为透明对象；反之，表示 TToolBar 对象为非透明对象，如图 6.12 所示。

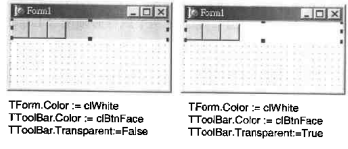


图 6.12　TToolBar 对象 Transparent 属性值

Wrapable 属性

Wrapable 属性是用来决定当 TToolBar 对象上的 TToolButton 对象若超过窗口宽度的处理方式。当 Wrapable 属性值为 True，表示当 TToolButton 对象若超过窗口宽度时会自动往下排列；反之，当 Wrapable 属性值为 False 时，则表示当 TToolButton 对象若超过窗口宽度时并不会往下排列，如图 6.13 所示。

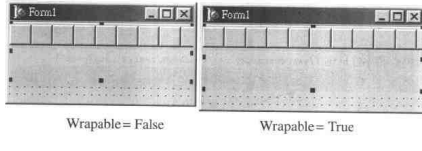
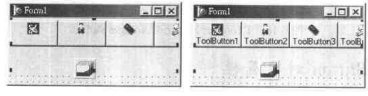


图 6.13　TToolBar 对象 Wrapable 属性值

ShowCaptions 属性

ShowCaptions 属性决定是否显示 TToolButton 对象上的文字(Caption 属性)。假如 ShowCaptions 属性值为 True 时，若 TToolButton 对象没有特别指定 Capiton 属性，则将 TToolBar 的 Capiton 属性值显示在 TToolButton 对象上；反之，若 TToolButton 对象特别指定 Capiton 属性，则直接显示 TToolButton 对象的 Caption 属性。当然，如果您将 ShowCaptions 属性值设为 False 时，则不显示任何文字在 TToolButton 对象，如图 6.14 所示。

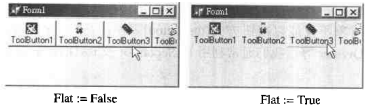


ShowCaptions:= False　　ShowCaptions:= True

图 6.14　TToolBar 对象 ShowCaptions 属性值

Flat 属性

Flat 属性主要是决定 TToolButton 对象是否可以成为一种平坦的显示方式(显示与窗口一体成形的感觉，当光标指向按钮时，按钮才会“浮”出来，如图 6.15 右图所示)。当 Flat 属性值为 True 时，表示 TToolButton 对象是一种平坦的按钮(如 Microsoft Word 2000 的按钮)。反之，当 Flat 属性值为 False 时，则表示 TToolButton 对象是一般的按钮对象。



Flat := False　　Flat := True

图 6.15　TToolBar 对象 Flat 属性值

DisableImages 属性

DisableImages 属性是用来决定当 TToolButton 对象的 Enable 属性被设为 False 时，所对应图片列表(ImageList)，这个属性值必须指定为一个 TImageList 对象名称，且配合 Flat 属性为 True 一起使用。

HotImages 属性

HotImages 属性是用来决定当用户将鼠标指定某 TToolButton 对象，所对应的图片列表(ImageList)，这个属性值必须指定为一个 TImageList 对象名称，且配合 Flat 属性为 True 一起使用。

Images 属性

Images 属性是用来决定 TToolButton 对象所对应的图片列表(ImageList)，这个属性值必须指定为一个 TImageList 对象名称，且配合 Flat 属性为 True 一起使用，如图 6.16 所示。



图 6.16 TToolBar 对象各项 Images 属性值

6.3.2 *TToolButton* 对象属性

前面曾经说明 TToolBar 对象是由 TToolButton 对象所组成的。TToolButton 对象事实上和 TBitButton 对象非常类似，但惟一的不同是 TToolButton 对象比 TBitButton 对象更酷。其实说穿了，TToolButton 对象必须靠 TToolBar 的 Flat 属性才能酷得了，否则与 TBitButton 对象没什么两样。虽是这么说，TToolButton 对象也是有其特别的属性，分述如下：

Wrap 属性

Wrap 属性是决定某个 TToolButton 对象是否折到下一行排列。此属性必须配合 TToolBar 对象的 Wrapable 属性一起使用。

Style 属性

TToolButton 对象不仅能以按钮方式(tbsButton)显示，另有 tbsCheck、tbsDropDown、tbsSeparator 及 tbsDivider 等。我们通常利用 tbsSeparator 及 tbsDivider 将 TToolButton 加以分隔。

范例 6-4

目的：本范例主要示范如何使用 TToolBar 对象产生一个类似于 Microsoft Internet Explorer(平坦式)的工具栏。

Step 1 创建新的工程。

Step 2 在 TForm 对象上加入两个 TImageList 对象及一个 TToolBar 对象。并将其属性设置成表 6.4 所示。

表 6.4

对象	属性名称	属性设置值
TForm(Form1)	WindwoState	wsMaximized
TImageList	Name	ImageList
TImageList	Name	HotImageList

续上表

对象	属性名称	属性设置值
TToolBar	Name	ToolBar1
	Images	ImageList
	HotImages	HotImageList
	Flat	True
	ShowCaptions	True

3 为 ImageList 对象加入图片文件。在 ImageList 对象上双击鼠标右键，此时会出现“ImageList”对话框，接着单击“Add”按钮，为 ImageList 加入图片文件(本范例的图形文件取自 C:\Program Files\Common Files\Borland Shared\Images\ Buttons 目录中，这些图片都包含彩色与黑白两种样式，我们在 ImageList 对象中使用黑白的部分，而 HotImageList 则使用彩色的部分)，如图 6.17 所示。

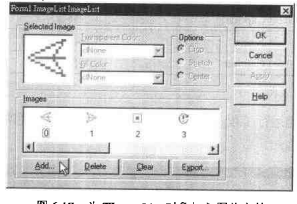


图 6.17　为 TImageList 对象加入图片文件

4 照着上一个步骤为另一个 TImageList 对象 HotImageList 对象加入图片文件。

5 在 TToolBar 对象上按右键，选择“New Button”，加入 10 个 TToolButton。由于在前面两个步骤已经加入了两个 TImageList 对象，已指定其与 TToolBar 对象之间的关系，因此在这里只要您加入新的按钮，系统会自动为这个新的按钮指定 Image 属性值，但是您也可以自行通过 ImageIndex 属性进行调整。这 10 个 TToolButton 对象的属性值设置如表 6.5 所示。

表 6.5

对象	属性名称	属性设置值
TToolButton	Name	ToolButton1
	Caption	后退
TToolButton	Name	ToolButton2
	Caption	前进
TToolButton	Name	ToolButton3
	Caption	停止

续上表

对象	属性名称	属性设置值
TToolButton	Name	ToolButton4
	Caption	刷新
TToolButton	Name	ToolButton5
	Caption	主页
TToolButton	Name	ToolButton6
	Caption	搜索
TToolButton	Name	ToolButton7
	Caption	收藏夹
TToolButton	Name	ToolButton8
	Caption	打印
TToolButton	Name	ToolButton9
	Caption	字体
TToolButton	Name	ToolButton10
	Caption	邮件

⑥ 保存工程。

⑦ 按【F9】功能键运行工程。工程运行后，您是不是发现所您制作的工具栏非常的酷！如图 6.18 所示。



图 6.18 自己制作的 TToolBar 工具栏

6.4 TCoolBar 对象

TCoolBar 对象也是工具栏的一种，位于对象面板上的 Win32 页标签中。TCoolBar 对象含有能够独立移动或调整大小的栏(Band)。这些个别的 Band，可以放置各种不同的控件。在实现上，我们通常会将第 6.3 节所介绍的 TToolBar 对象加入 TcoolBar 对象中的某个 Band。以下将对 TcoolBar 对象几个常用且很重要的属性加以说明。

6.4.1 TCoolBar 对象属性

Bands 属性

Band 属性用来存放一组 TCoolBand 对象。在设计时期可以通过 Bands Editor 来新增、修

改或删除某个 Band。有关 TCoolBand 对象，请参考第 6.4.2 节介绍。

- *BandBorderStyle 属性*

BandBorderStyle 属性是用来决定 Band 是否有边框，当 BandBorderStyle 属性值为 bsNone 表示 Band 没有边框；反之，当 BandBorderStyle 属性值为 bsSingle 表示 Band 拥有边框。

- *BorderWidth 属性*

BorderWidth 属性是用来决定 TCoolBand 对象与 TCoolBar 对象之间的距离。

- *FixedOrder 属性*

FixedOrder 属性是用来决定用户在运行期间是否可以重新排列工具栏。

- *FixedSize 属性*

FixedSize 属性是用来决定在 TcoolBar 对象中的 Band 是否要自动保持一致的高度与宽度。

- *Bitmap 属性*

Bitmap 属性是用来决定 TcoolBar 对象上所有 TCoolBand 的背景图片。您可以在 Bitmap 属性字段上双击打开“Picture Editor”来选择指定的背景图片文件。

- *ShowText 属性*

ShowText 属性决定是否在 TCoolBand 对象中显示 Text 属性内容，假如 ShowText 为 True，则每一个 TCoolBand 对象的 Text 内容在运行时会被显示出来。

6.4.2　TCoolBand 对象属性

- *Bitmap 属性*

Bitmap 属性是用来决定 TcoolBand 对象的背景图片。您可以在 Bitmap 属性字段上双击鼠标左键打开 Picture Editor 来选择指定的背景图片文件。它与 TCoolBar 对象的 Bitmap 属性最大的不同在于 TCoolBar 对象的 Bitmap 属性是针对所有的 TCoolBand，而 TCoolBand 对象上的 Bitmap 是针对个别的 TCoolBand 对象。

- *Control 属性*

Control 属性是用来决定某个 TCoolBand 对象所控制的其他对象，如 TToolBar、TcomboBox、TEdit 等。几乎大部分的可见(visible)对象都是 TCoolBand 可控制对象。

- *ImageIndex 属性*

ImageIndex 属性是用来决定这个 TCoolBand 的图片，ImageIndex 属性必须在 TCoolBar 对象的 Images 属性指定某个 TImageList 对象才会有效。如果在 TCoolBar 对象的 Images 属性

指定了某个 TImageList 对象时，您只要在 ImageIndex 属性输入这个 TImageList 对象中的图片编号即可。

Text 属性

Text 属性是用来设置 TCoolBand 对象所显示的提示文本。Text 属性必须配合 TCoolBar 对象 ShowText 属性值为 True 才会有效。

ParentBitmap 属性

ParentBitmap 属性是用来决定此 TCoolBand 对象是否与其所属的 TCoolBar 对象共享背景图片。ParentBitmap 属性必须配合 TCoolBar 对象 Bitmap 属性值指定一个位图文件才会有效。

范例 6-5

目的：本范例主要示范如何使用 TCoolBar 对象（见图 6.19）。

STEP 1 本范例主要示范如何利用 TCoolBar 对象集成其他工具栏或对象，因此为节省 TToolBar 工具栏的制作时间，故修改自范例 6-4。

STEP 2 将范例 6-4 中所创建的 TToolBar 对象的 Transparent 属性设置 True。

STEP 3 在 TForm 对象中加入一个 TCoolBar 对象及一个 TComboBox 对象，其属性设置如表 6.5 所示。

表 6.5

对象	属性名称	属性设置值
TCoolBar	Name	CoolBar1
	Bitmap	/范例 6-5/bg.bmp
	AutoSize	True
TComboBox	Name	ComboBox1

STEP 4 在 TCoolBar 对象中加入两个 TCoolBand，其属性设置如表 6.6 所示。

表 6.6

对象	属性名称	属性设置值
TCoolBand	Name	CoolBand1
	Control	ToolBar1
TCoolBand	Name	CoolBand2
	Control	ComboBox1

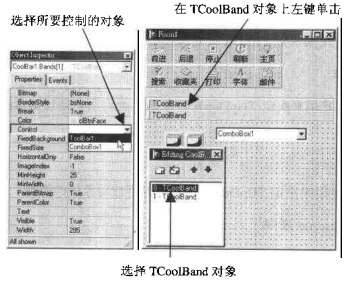


图 6.19　设置 TCoolBand 对象属性

5　保存文件。

6　按【F9】键运行工程，此时您可以任意移动 CoolBand1 及 CoolBand2 的顺序。除此之外，由于我们将 TToolBar 对象设为透明对象，因此会出现在 TCoolBar 对象所设置的背景图片。

6.5　*TUpDown 与 TSpinEdit 对象*

TUpDown 对象，位于对象面板上的 Win32 页标签，可以连接 TEdit 或其他对象，并且由 TUpDown 对象来控制被连接对象的值。其用途与 TSpinEdit 对象相当的类似，您可以将 TSpinEdit 对象(位于对象面板上的 Sample 页标签)想象为 TUpDown 对象与 TEdit 对象结合的特例。以下就 TUpDown 对象各个特殊属性加以说明。

Orientation 属性

Orientation 属性是用来决定 TUpDown 对象显示的类型，有水平显示(udHorizontal)及垂直(udVertical)显示两种，如图 6.20 所示。

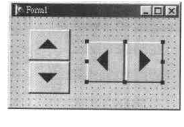


图 6.20　TUpDown 对象 Orientation 属性

- *Associate* 属性

Associate 属性是用来决定 TUpDown 对象所要连接的对象。

- *ArrowKeys* 属性

ArrowKeys 属性是用来决定 TUpDown 对象是否允许用户通过键盘的上、下方向键来改变所连接对象的值。

- *Increment* 属性

Increment 属性是用来决定 TUpDown 对象对所连接对象的改变量。

- *Max*、*Min*、*Position* 属性

Max 属性是 TUpDown 对象允许用户的最大输入值，Min 属性是 TUpDown 对象允许用户的最小输入值，Positon 属性是 TUpDown 对象目前的值，

范例 6-6

目的：本范例主要示范如何使用 TUpDown 对象及 TSpinEdit 对象。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TUpDown 对象、一个 TEdit 对象及一个 TSpinEdit 对象，其相关属性设置如表 6.7 所示。

表 6.7

对象	属性名称	属性设置值
TEdit	Name	Edit1
TUpDown	Name	UpDown1
	Associate	Edit1
TSpinEdit	Name	SpinEdit1

(3) 保存文件。

(4) 按【F9】键运行工程，您可以通过鼠标或键盘改变 TUpDown 对象所连接的对象值或 TSpinEdit 对象，如图 6.21 所示。

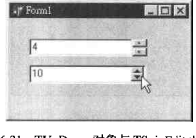


图 6.21 TUpDown 对象与 TSpinEdit 对象

6.6　TSpeedButton 对象

TSpeedButton 对象与 TToolBar 对象在功能上非常类似，位于对象面板上的 Additional 页标签中。在实现上我们通常会配合 TPanel 对象一起使用。以下就 TSpeedButton 对象的各项常用属性加以说明：

- *Flat 属性*

Flat 属性是用来决定 TSpeedButton 对象是否具有 Flat 的特性。与 TToolBar 对象最大的不同在于 TToolBar 对象的 Flat 属性针对所有的 TToolButton 对象，而 TSpeedButton 对象是针对各别的对象。

- *Glyph 属性*

Glyph 属性是用来设置 TSpeeedButton 对象的图标文件。

- *Layout 属性*

Layout 属性主要是决定 TSpeedButton 对象中图标(BitMap)与文本说明的排列方式，Layout 属性共有 blGlyphLeft、blGlyphRight、blGlyphTop 及 blGlyphBottom 等四种。请参考第 6.1 节 TbitButton 对象的详细说明。

- *Transparent 属性*

Transparent 属性是用来决定 TSpeedButton 对象是否为透明对象。

- *GroupIndex 属性*

GroupIndex 属性是用来决定哪些 TSpeedButton 对象为相同的群组。如果您要将某些 TSpeedButton 对象设置同一组，只要将 GroupIndex 属性值指定为相同的非零整数即可。

- *Down 属性*

Down 属性是用来决定某个 TSpeedButton 对象是否为被选取状态。

- *AllowAllUp 属性*

如果您设置某一个 TSpeedButton 对象的 AllowAllUp 属性值，则 Delphi 6 会自动将同一组的所有 TSpeedButton 对象设为相同属性值。当 AllowAllup 属性值为 True 时，表示在同一组中的 TSpeedButton 对象都可同时为不被选取的状态；反之，如果 AllowAllUp 属性值为 False 时，则表示在同一组中至少要有一个 TSpeedButton 对象为被选取状态。

范例 6-7

目的：本范例主要示范如何配合 TPanel 对象使用 TSpeedButton 对象。

Step 1 创建新的工程。

Step 2 在 TForm 对象上加入一个 Tpanel 对象，并将其 Align 属性设置 alTop，Height 属性设为 37。

Step 3 在 Tpanel 对象上加入两个 TLabel 对象、四个 TSpeedButton、一个 TEdit 对象及一个 TUpDown 对象。其相关属性设置如表 6.7 所示。

表 6.7

对象	属性名称	属性设置值
TEdit	Name	Edit1
	Text	1
TUpDown	Name	UpDown1
	Associate	Edit1
	Min	1
TLabel	Name	Label1
	Caption	画笔颜色
TLabel	Name	Label2
	Caption	画笔大小
TSpeedButton	Name	SpeedButtonRed
	Caption	红色
	AllowAllUp	False
	GroupIndex	1
TSpeedButton	Name	SpeedButtonBlack
	Caption	黑色
	AllowAllUp	False
	GroupIndex	1
TSpeedButton	Name	SpeedButtonBlue
	Caption	蓝色
	AllowAllUp	False
	GroupIndex	1
TSpeedButton	Name	SpeedButtonClose
	Caption	结束
	Glyph	/CommandFiles/Borland Shared/Images/Icons/chem16.bmp

Step 4 在 SpeedButtonRed 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.SpeedButtonREDClick(Sender: TObject);
begin
    //设置画笔颜色(红色)
    Canvas.Pen.Color:=clRed;
```

```
end;
```

STEP 5 在 SpeedButtonBlack 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.SpeedButtonBlackClick(Sender: TObject);
begin
    //设置画笔颜色(黑色)
    Canvas.Pen.Color:=clBlack;
end;
```

STEP 6 在 SpeedButtonBlue 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.SpeedButtonBlueClick(Sender: TObject);
begin
    //设置画笔颜色(蓝色)
    Canvas.Pen.Color:=clBlue;
end;
```

STEP 7 在 SpeedButtonClose 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.SpeedButtonCloseClick(Sender: TObject);
begin
    Close; //结束工程
end;
```

STEP 8 在 TEdit 对象的 onChange 事件处理过程中加入以下的代码：

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  //改变画笔宽度
  Canvas.Pen.Width:=StrToInt(Edit1.Text);
end;
```

STEP 9 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    //设置画图状态
    DrawFlag:=False;
    //设置画笔颜色
    SpeedButtonRed.Down:=True;
    Canvas.Pen.Color:=clRed;
end;
```

STEP 10 在 TForm 对象的 onMouseDown 事件处理过程中加入以下的代码：

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
  //记录第一点坐标
```

```
  FirstX:=X;
  FirstY:=Y;
  //进入画图状态
  DrawFlag:=True;
end;
```

(11) 在 TForm 对象的 onMouseMove 事件处理过程中加入以下的代码：

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
Begin
  if DrawFlag then
    begin
      //由第一点画出一条直线
      Canvas.MoveTo(FirstX,FirstY);
      Canvas.LineTo(X,Y);
      //改变第一点坐标
      FirstX:=X;
      FirstY:=Y;
    end;
end;
```

(12) 在 TForm 对象的 onMouseUp 事件处理过程中加入以下的代码：

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;  Shift:
TShiftState; X, Y: Integer);
begin
  //离开画图状态
  DrawFlag:= NOT DrawFlag;
end;
```

(13) 保存文件。

(14) 按【F9】功能键运行工程，工程运行后，您可以改变画笔颜色及画笔宽度，并画出所要的图案，如图 6.22 所示。

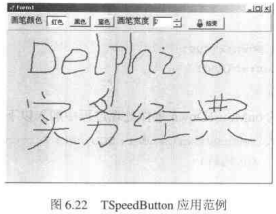


图 6.22　TSpeedButton 应用范例

6.7　TSpinButton 对象

TSpinButton 对象与 TUpDown 对象有些类似但不完全相像，位于对象面板上的 Sample 页标签中。您可以想象 TSpinButton 对象是由两个 TSpeedButton 组合而成。TSpinButton 对象与其他的按钮对象一样都有 Name、Caption、Left、Top、Width 及 Height 等的属性。这里笔者仅介绍 DownGlyph 属性、UpGlyph 属性、onDonwClick 事件与 onUpClick 事件等。

- *DownGlyph* 属性

您可以通过 TSpinButton 对象的 DownGlyph 属性更改 Down 按钮的默认小图标 ˄ 。

- *UpGlyph* 属性

您可以通过 TSpinButton 对象的 UpGlyph 属性更改 Up 按钮的默认小图标 ˅ ，如图 6.23 所示。

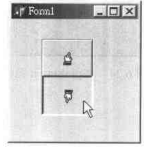


图 6.23　更改 TSpinButton 小图标

- *OnDonwClick* 事件

当 Down 按钮被鼠标左键按下时 onDonwClick 事件即被触发。

- *onUpClick* 事件

当 Up 按钮被鼠标左键按下时 onUpClick 事件即被触发。

范例 6-8

目的：本范例主要的目的是示范如何使用 TSpinButton 对象，您可以与范例 6-6 进行比较。

1. 创建新的工程。
2. 在 TForm 对象上加上一个 TEdit 与一个 TSpinButton 对象，其摆放位置如图 6.24 所示，并设置 TEdit 对象的 Text 属性值为 0。
3. 在 TSpinButton 对象的 onDownClick 事件处理过程中加入以下的代码，此代码主要功能是将 Edit1 的 Text 属性值减 1。

```
procedure TForm1.SpinButton1DownClick(Sender: TObject);
```

```
begin
    Edit1.Text := IntToStr(StrToInt(Edit1.text) - 1);
end;
```

④ 在 TSpinButton 对象的 onUpClick 事件处理过程中加入以下的代码，此代码主要功能是将 Edit1 的 Text 属性值加 1。

```
procedure TForm1.SpinButton1UpClick(Sender: TObject);
begin
    Edit1.Text := IntToStr(StrToInt(Edit1.text) + 1);
end;
```

⑤ 保存并运行工程。工程运行后，您就可以像范例 6-6 所示范的一样任意增加或减少 Edit1 的 Text 属性值，如图 6.24 所示。

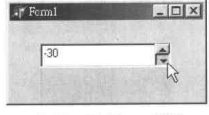


图 6.24　TSpinButton 范例

第7章

Delphi 6 文本控制对象

本章目标

在应用程序开发中最重要的一件事，就是要决定如何与用户沟通。上一个章节已经介绍 Delphi 6 所提供的按钮对象，本章接着为您介绍 Delphi 6 所提供的文本控制对象，包括 TEdit、TMaskEdit、TMemo、TRichEdit、TLable、TStringGrid 及 TLabeledEdit 对象。

本章主题

7.1 Tedit 对象

7.2 TMaskEdit 对象

7.3 TMemo 对象

7.4 TRichEdit 对象

7.5 TLabel 对象

7.6 TStringGrid 对象

7.7 TLabeledEdit 对象

在 Delphi 6 文本控制对象中，同时具有输入与输出功能的对象包括 TEdit、TMaskEdit、TMemo 及 TRichEdit 等。本小节先说明这些对象共同的属性。其余特有属性在其他小节另有详细说明。以下说明 TEdit、TMaskEdit、TMemo 及 TRichEdit 等对象的特性及使用时机。

- *TEdit 对象*

TEdit 对象只能编辑一行的文本，没有字符容量的限制，但通常用于少量文本的输入输出接口，例如：用户的帐号或用户的密码等。TEdit 对象位于组件面板的 Standard 页标签中。

- *TMaskEdit 对象*

TMaskEdit 对象与 TEdit 对象同样是只能编辑一行的文本，故也是用于少量文本的输入输出接口。它最大的特点是可自行定义特定的格式，以强迫用户输入。例如电话号码、生日、统一编号等特定格式。TMaskEdit 对象位于组件面板的 Additional 页标签中。

- *TMemo 对象*

您可以假想 TMemo 对象是由多个 TEdit 对象所组合，它可以编辑多行的文本。TMemo 对象位于组件面板的 Standard 页标签中。

- *TRichEdit 对象*

您可以将 TRichEdit 对象想象为 TMemo 对象的豪华版，它支持 RTF 格式的 TMemo 对象，TRichEdit 可以编辑多行的文本，而其容量基本上是没有限制的。它最大的特点就是可以对对象中的文本分别改变字体、格式及拖动等简单的文字处理功能。TRichEditt 对象位于组件面板的 Win32 页标签中。

7.1 TEdit 对象

TEdit 对象位于组件面板的 Standard 页标签中，它只能编辑一行的文本，故通常用于少量文本的输入输出接口，例如：用户的帐号或用户的密码等。表 7.1 将其属性(Properties)及表 7.2 操作方法(Methods)加以列表说明，基于本篇幅及使用频率的考虑，我们并不会对所有的属性及操作方法加以一一详述，我们仅挑选出最常使用的属性及操作方法加以说明。(注：加下划线的属性代表本章节会有详细说明的属性)

表 7.1 TEdit 对象的属性

属性(Properties)	▲表示执行时期才能使用	
AutoSelect	AutoSize	BorderStyle
▲CanUndo	CharCase	HideSelection
MaxLength	Modified	OEMConvert
PasswordChar	ReadOnly	SelLength
SelStart	SelText	▲Brush
▲ClientOrigin	▲ClientRect	▲ControlCount

续上表

属性(Properties)	▲表示执行时期才能使用	
▲Controls	Ctl3D	▲Handle
HelpContext	ImeMode	ImeName
ParentCtl3D	ParentWindow	▲Showing
TabOrder	TabStop	Align
Anchors	BiDiMode	BoundsRect
ClientHeight	ClientWidth	Color
Constraints	ControlState	ControlStyle
Cursor	DragCursor	DragKind
DragMode	Enabled	Font
Height	Hint	Left
Name	Parent	ParentBiDiMode
ParentColor	ParentFont	ParentShowHint
PopupMenu	ShowHint	Text
Top	Visible	Width
WindowProc		

表 7.2

操作方法(Methods)		
Clear	ClearSelection	ClearUndo
CopyToClipboard	Create	CutToClipboard
GetSelTextBuf	PasteFromClipboard	SelectAll
SetSelTextBuf	Undo	Broadcast
CanFocus	ContainsControl	ControlAtPos
CreateParented	Destroy	DisableAlign
EnableAlign	Focused	GetTabOrderList
HandleAllocated	HandleNeeded	InsertControl
Invalidate	PaintTo	Realign
RemoveControl	Repaint	ScaleBy
ScrollBy	SetBounds	SetFocus
Update	UpdateControlState	BeginDrag
BringToFront	ClientToScreen	DragDrop
Dragging	EndDrag	GetTextBuf
GetTextLen	Hide	Perform
Refresh	ScreenToClient	SendToBack
SetTextBuf	Show	

7.1.1 TEdit 对象的属性

以下将 TEdit 对象常用的属性分别说明如下：

- *Text 属性*

Text 属性是获取或设置 TEdit 对象的文本内容。

- *MaxLength 属性*

MaxLength 属性是用来设置 TEdit 对象最长的可输入字符。系统默认值为 0，表示没有限制最长的输入字符。您可以根据输入数据的需求自行设置合理的最长字符串限制，以维持数据的有效性。

- *CharCase 属性*

CharCase 属性是强迫将用户所输入的文本转为大写(ecUpperCase)、小写(ecLowerCase)或两者混合(ecNormal)。

- *PasswordChar 属性*

PasswordChar 属性通常是用于隐藏用户所输入的文本内容，例如用户所输入的密码。PasswordChar 默认值为#0，表示不作任何改变，#0 也就是空字符串的意思。如果您要指定特别字符，可以直接输入即可。在习惯上，密码输入我们会用“*”取代。



目的：本范例通过“密码确认对话框”的制作，来说明 TEdit 对象的 Text、MaxLength、CharCase 及 PasswordChar 等属性的使用方法。

1. 创建一个新的工程。
2. 在默认的 TForm 对象中加入两个 TLabel、两个 TEdit 及两个 TButton 对象，其属性设置如表 7.3 所示（对象摆放位置，请参见图 7.1 所示）。

表 7.3

对象 (name 属性)	属性名称	属性设置值
TLabel(Label1)	Caption	请输入您的帐号
TLabel(Label2)	Caption	请输入您的密码
TEdit(EditID)	Text	(清除 Text 属性值)
	CharCase	ecUpperCase
	MaxLength	10
TEdit(EditPWD)	Text	(清除 Text 属性值)
	PasswordChar	$
TButton(Button1)	Caption	确定
TButton(Button2)	Caption	取消



图 7.1　密码对话框

STEP 3　在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //显示用户的帐号及密码
  ShowMessage('您所输入的帐号是：' +EditID.Text+
                          #10+'您所输入的密码是：'+EditPwd.Text);
end;
```

STEP 4　在 Button2 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //清除用户所输入的帐号及密码
  EditID.Text:='';
  EditPwd.Text:='';
end;
```

STEP 5　保存工程。

STEP 6　按【F9】功能键运行程序。此时您所输入的用户帐号会自动变为大写，而且有 10 个字的最大长度限制。另外，您所输入的密码会自动以“S”加以取代，以达保密性。最后当您单击“确定”按钮，系统会显示出您所输入的帐号与密码，如图 7.1 所示。

ReadOnly 属性

ReadOnly 属性是用来指明是否允许用户对 TEdit 对象进行输入的操作，当 ReadOnly 属性值为 True，表示此 TEdit 对象仅供输出。反之，则表示可为输出与输入。ReadOnly 属性值并不会影响运行时期程序的直接设置，换言之，当 TEdit 对象的 ReadOnly 属性值为 True，我们在程序中同样可以指定其 Text 的内容值。

ImeName 属性

ImeName 属性允许用户为此 TEdit 对象设置特定的中文输入法，例如您将 ImeName 属性值设置“全拼”(注意：您的计算机系统必须要存在此输入法)，当此 TEdit 对象获取焦点时，输入法会自动切换成“全拼”中文输入法。在这里您必须特别注意一点，如果您已经指定 PasswordChar 属性的特别值时，您是无法输入中文本的。

Modified 属性

Modified 属性是用来记录此 TEdit 对象 text 属性值是否被修改过。

范例 7-2

目的：本范例主要目的在说明 TEdit 对象的 ReadOnly、ImeName 及 Modified 等属性的使用方法。

1. 创建新的工程。
2. 在 TForm 对象中加入两个 TLabel 及两个 TEdit 对象，其属性设置如表 7.4 所示（对象摆放位置，请参见图 7.2 所示）。

表 7.4

对象 (name 属性)	属性名称	属性设置值
TLabel(Label1)	Caption	可以输入的 TEdit
TLabel(Label2)	Caption	只读的 TEdit
TEdit(Edit1)	ImeName	智能 ABC
TEdit(Edit2)	ReadOnly	True

3. 在 TForm 对象的 onCloseQuery 事件处理过程中加入以下的代码。当您关闭 TForm 对象，同时会触发 OnCloseQuery 事件，因此可以通过此事件来显示用户是否曾经更改过 TEdit1 的内容。

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if Edit1.Modified then
    ShowMessage('Edit1 的内容已经被修改....');
end;
```

4. 保存工程。
5. 按【F9】功能键运行工程。程序运行后，当您将焦点移到 TEdit1 时，会自动打开全拼输入法，但无法编辑 TEdit2 的内容。当您结束工程时，程序会判断您是否曾经对 TEdit1 加以修改，如图 7.2 所示。

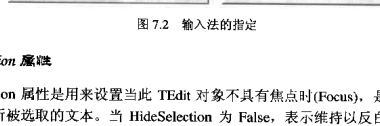


图 7.2 输入法的指定

HideSelection 属性

HideSelection 属性是用来设置当此 TEdit 对象不具有焦点时(Focus)，是否维持以反白方式来显示目前所被选取的文本。当 HideSelection 为 False，表示维持以反白方式来显示。反之，则取消反白。

- *SelLength* 属性

SelLength 属性是用来获取用户所选取的字符串长度。

- *SelStart* 属性

SelStart 属性是用来获取用户所选取字符串的起始位置。

- *SelText* 属性

SelText 属性是用来获取用户所选取的字符串内容。

范例 7-3

目的：本范例主要目的是说明 TEdit 对象的 HideSelection、SelLength、SelStart 及 SelText 等属性的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象加入两个 TEdit 对象及一个 TButton 对象，其属性值设置如表 7.5 所示(对象摆放位置，请参见图 7.3 所示)。

表 7.5

对象名称	属性名称	属性设置值
TEdit	Name	Edit1
	HideSelection	False
TEdit	Name	Edit2
TButton	Name	Button1
	Caption	显示 Edit1 的选取属性值

STEP 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Str:String;
begin
   Str:='您所选取的字符串长度:' +IntToStr (Edit1.SelLength)
        +#10;
   Str:=Str+'您所选取的字符串起始位置:'+IntToStr
        (Edit1.SelStart ) +#10;
   Str:=Str+'您所选取的字符串:'+Edit1.SelText;
   ShowMessage(Str);
end;
```

STEP 4 保存工程。

STEP 5 按【F9】功能键运行工程。工程运行后，您可以同时选取 Edit1 及 Edit2 的部分字符

串。您也可以利用按下 Button1 来查看 Edit1 目前有关选取的属性值，如图 7.3 所示。



图 7.3 字符串选取相关属性

- *CanUndo* 属性

CanUndo 属性值用来记录此 TEdit 对象目前是否具有 Undo 的功能，在您改变 TEdit 对象的内容之前，CanUndo 属性值通常为 False，在您改变 TEdit 对象的内容之后，CanUndo 属性值则 True。CanUndo 是一个只读的属性，您无法去更改它的值，通常配合 ClearUndo 及 Undo 等操作方法使用(详细用法请参考 ClearUndo 及 Undo 等操作方法)。

7.1.2 TEdit 对象的操作方法

以下将 TEdit 对象常用的操作方法属性分别说明如下：

- *Clear* 操作方法

Clear 操作方法是用来清除 TEdit 对象的内容，也就是删除 Text 属性值。

- *SelectAll* 操作方法

SelectAll 操作方法可以选取 TEdit 对象中所有的文本内容，必须配合 HideSelection 属性值为 False 使用。

- *ClearSelection* 操作方法

ClearSelection 操作方法是用来删除 TEdit 对象中被选取的内容，也就是指删除 SelText 属性值。

范例 7-4

目的：本范例主要的目的是说明如何使用 TEdit 对象的 Clear、SelectAll 及 ClearSelection 等操作方法。

(1) 创建新的工程。

(2) 在工程中加入一个 TEdit 对象及三个 TButton 对象。其属性值设置如表 7.6 所示 (对象摆放位置，请参见图 7.4 所示)。

表 7.6

对象 (name 属性)	属性名称	属性设置值
TEdit(Edit1)	HideSelection	False
TButton(Button1)	Caption	Clear
TButton(Button2)	Caption	SelectAll
TButton(Button3)	Caption	ClearSelection

(3) 分别在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Clear;
end;
```

(4) 分别在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.SelectAll;
end;
```

(5) 分别在 Button3 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit1.ClearSelection;
end;
```

(6) 保存工程。

(7) 按【F9】功能键运行工程，此时您可以利用 Clear 按钮来清除 Edit1 对象的所有内容；您也可以利用 SelectAll 按钮来选取所有的文本，再利用 ClearSelection 按钮来清除所选取的字符串，如图 7.4 所示。

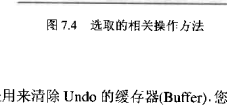


图 7.4　选取的相关操作方法

ClearUndo 操作方法

ClearUndo 操作方法是用来清除 Undo 的缓存器(Buffer)，您可以使用 ClearUndo 去确认您对 TEdit 对象所作的改变。ClearUndo 此操作方法会将用户所作的改变恢复到上一次运行 ClearUndo 操作方法的结果。

Undo 操作方法

Undo 操作方法可以复原用户对 TEdit 对象所作的改变，在您没有使用 ClearUndo 的操作方法之前，您可以利用 Undo 此操作方法将改变的地方复原到原来的状态。

范例 7-5

目的：本范例主要是说明如何使用 TEdit 对象的 ClearUndo 及 Undo 等操作方法，本范例配合 TEdit 对象 CanUndo 属性使用。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TEdit 及三个 TButton 对象，其属性值设置如表 7.6 所示(对象摆放位置，请参见图 7.5 所示)。

表 7.6

对象 (name 属性)	属性名称	属性设置值
TEdit(Edit1)		
TButton(Button1)	Caption	显示 CanUndo 属性
TButton(Button2)	Caption	Undo
TButton(Button3)	Caption	ClearUndo

STEP 3 在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Edit1.CanUndo then
    ShowMessage('Edit1.CanUndo=True')
  else
    ShowMessage('Edit1.CanUndo=False');
end;
```

STEP 4 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Undo;
end;
```

STEP 5 在 Button3 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit1.ClearUndo;
end;
```

STEP 6 保存工程。

7 按【F9】键运行工程。工程运行后，Edit1 的 CanUndo 属性值默认置 False，那是因为 Edit1 尚未有任何的改变。您可以利用 Undo 操作方法将改变的地方恢复到原来的状态。也可以用 ClearUndo 去确认您对 TEdit 对象所作的改变，如图 7.5 所示。

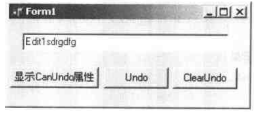


图 7.5　编辑的复原

- *CopyToClipboard* 操作方法

CopyToClipboard 操作方法是将指定内容复制到系统剪贴板中。

- *CutToClipboard* 操作方法

CutoClipBorad 操作方法是将指定的内容移动到系统的剪贴板中。

- *PasteFromClipboard* 操作方法

PasteFromClipboard 操作方法是将系统剪贴板的内容粘贴到指定对象中。

范例 7-6

目的：大部分的文字处理工具都有提供用户能够在文件中或文件之间对文本的处理功能，这通常必须借助于操作系统中的剪贴板(Clipborad)，本范例便是利用剪贴板的特性来完成简单的复制及替代的功能。

1 创建新的工程。

2 在 TForm 对象中加入一个 TEdit 及三个 TButton 对象，其属性值设置如表 7.8 所示(对象摆放位置，请参见图 7.6 所示)。

表 7.8

对象(name 属性)	属性名称	属性设置值
TEdit(Edit1)	HideSelection	False
TButton(Button1)	Caption	复制
Tbutton(Button2)	Caption	剪切
Tbutton(Button3)	Caption	粘贴

3 在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //复制选取文件到剪贴板
```

```
    Edit1.CopyToClipboard;
  end;
```

STEP 4 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    //将选取文本剪切到剪贴板
    Edit1.CutToClipboard;
  end;
```

STEP 5 在 Button3 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.Button3Click(Sender: TObject);
  begin
    //将剪贴板的内容粘贴到 Edit1
    Edit1.PasteFromClipboard;
  end;
```

STEP 6 保存工程。

STEP 7 按【F9】功能键运行工程。工程运行之后您就可以使用文字处理工具中的“复制”、“剪切”及“粘贴”等功能，如图 7.6 所示。

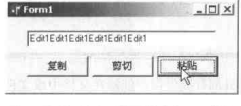


图 7.6 复制、剪切及粘贴等文字处理功能

- *GetSelTextBuf* 操作方法

将 TEdit 对象被选择的字符串复制到指定的临时保存空间(用户自定，非系统剪贴板)。

- *SetSelTextBuf* 操作方法

将 TEdit 对象被选择的字符串以指定临时保存空间(用户者自定，非系统剪贴板)加以替换。

范例 7-7

目的：系统的剪贴板虽然很好用，但是却只有一个。换言之，您只能保有最后一份文本在剪贴板中，如果您不只是想要如此的话，那么您可以使用下面的方法替换系统剪贴板。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TEdit 及三个 TButton 对象，其属性值设置如表 7.9 所示(对象摆放位置，请参见图 7.7 所示)。

表 7.9

对象　(name 属性)	属性名称	属性设置值
TEdit(Edit1)	HideSelection	False
TButton(Button1)	Caption	复制
Tbutton(Button2)	Caption	剪切
Tbutton(Button3)	Caption	粘贴

STEP 3 在单元文件(unit)中声明以下的变量。

```
Var
  MyClipBorad:PChar;//自定义剪贴板
  StrSize:Integer;  //选取字符串的大小
```

STEP 4 在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //复制选取文件到剪贴板
  StrSize := Edit1.SelLength-1;
  GetMem(MyClipBorad,StrSize);
  Edit1.GetSelTextBuf(MyClipBorad,StrSize);
end;
```

STEP 5 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //将选取文本剪切到剪贴板
  Button1Click(nil);
  Edit1.ClearSelection;
end;
```

STEP 6 在 Button3 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  //将剪贴板的内容粘贴到 Edit1
  Edit1.SelText:=StrPas(MyClipBorad);
end;
```

STEP 7 保存工程。

STEP 8 按【F9】功能键运行工程，工程运行之后您就可以使用文字处理工具中的“复制”、“剪切”及“粘贴”等功能，如图 7.7 所示。



图 7.7 复制、剪切及粘贴等文字处理功能

7.1.3 TEdit 对象的事件处理

以下将 TEdit 对象常用的事件处理分别说明如下：

- *OnChange* 事件

OnChange 事件处理是当 TEdit 对象的内容被改变时所触发的事件。

- *OnEnter* 事件

OnEnter 事件处理是当 TEdit 对象获取焦点时所触发的事件。

- *OnExit* 事件

OnExit 事件处理是当 TEdit 对象失去焦点时所触发的事件。

范例 7-8

目的：TEdit 对象有很多的事件处理过程，但大部分的事件处理过程在前面的章节已经说明了，所以本范例主要是说明 OnChange、OnEnter 及 OnExit 等事件。

1. 创建新的工程。
2. 在工程中加入两个 TEdit 对象，其 Name 属性分别为 Edit1 及 Edit2。
3. 在 Edit1 的 onChange 事件处理过程中加入以下的代码：

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  //将 Edit2 的内容设置为 Edit1 的内容
  Edit2.Text:=Edit1.Text;
end;
```

4. 在 Edit1 的 onEnter 事件处理过程中加入以下的代码：

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
   //当 Edit1 获取焦点，则将字符串颜色改为红色
   Edit1.Font.Color:=clRED;
end;
```

5. 在 Edit1 的 onExit 事件处理过程中加入以下的代码：

```
procedure TForm1.Edit1Exit(Sender: TObject);
```

```
begin
    //当 Edit1 失去焦点，则将字符串颜色改为黑色
    Edit1.Font.Color:=clBlack;
end;
```

步骤 6　保存工程。

步骤 7　按【F9】功能键运行工程，工程运行后，只要 Edit1 获取焦点时，其字体颜色变为红色，离开时又恢复成黑色，同时只要您更改 Edit1 的内容，Edit2 的内容会和 Edit1 同步更新，如图 7.8 所示。

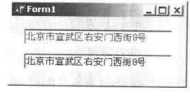


图 7.8TEdit 对象的事件处理过程

7.2　TMaskEdit 对象

TMaskEdit 对象与 TEdit 对象同时只能编辑一行的文本，故也是用于少量文本的输入输出接口。它最大的特点是可以自行定义特定的格式，以强迫用户输入特定格式的文本。例如电话号码、牛口、统一编号等特定格式。TMaskEdit 对象位于组件面板的 Additional 页标签中。由于 TMaskEdit 对象大部分的属性(Properties)及操作方法(Methods)都与 TEdit 对象相同，因此不再赘述，以下仅列出 TMaskEdit 对象特有且常用的属性与操作方法。

7.2.1　TMaskEdit 对象属性

EditMask 属性

EditMask 属性主要是用来设置掩码(Mask)的指定字符，换言之，我们可以利用 MaskEdit 对象所提供的 EditMask 属性强迫用户输入合法的字符。在实务上，我们会利用 Delphi 6 所提供的掩码编辑器(Input Mask Editor)来完成。

范例 7-9

目的：本范例主要是示范如何使用 Delphi 6 所提供的输入掩码编辑器(Input Mask Editor)来完成指定的输入格式。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TMaskEdit 对象(在 Additional 页标签)。

步骤 3　编辑 TMaskEdit 对象的 EditMask 属性值，本范例以时间格式为例，过程如图 7.9 所

示。在掩码编辑器中的“Character for Blanks”是指定在掩码中以什么字符来表示空格符。“Save Literal Characters”复选框(CheckBox)是用来决定是否将掩码当成是数据的一部分。

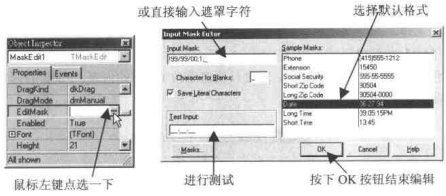


图 7.9　编辑 TMaskEdit 对象的 EditMask 属性值

4 保存工程。

5 按【F9】功能键运行工程，如图 7.10 所示。

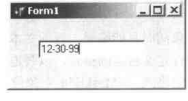


图 7.10　TMaskEdit 对象的示范

EditText 属性

EditText 属性主要是用来记录 TMaskEdit 对象的掩码(Mask)格式。这是一个只读属性，您在设计时期是无法加以修改的。以范例 7-9 为例，其 EditText 属性如图 7.9 所示，您只要写 ShowMessage (MaskEdit1.EditText)即可获取，如图 7.11 所示。



图 7.11　TMaskEdit 对象的 EditText 属性

IsMasked 属性

IsMasked 属性也是一个只读属性，它是用来记录此 TMaskEdit 对象是否设置了掩码。当 IsMasked 为 True，表示 TMaskEdit 对象已经设置了掩码字符。反之，则代表尚未设置。

Text 属性

Text 属性可以用来设置掩码字符中的分隔字符，此属性也适用于运行时期。以范例 7-9 为例，Text 属性值如图 7.12 所示。



图 7.12 TMaskEdit 对象的 Text 属性

范例 7-10

目的：本范例主要修改范例 7-9，在范例 7-9 中主要是在设计时期利用 Delphi 6 所提供的输入掩码编辑器(Input Mask Editor)来完成指定的输入格式。本范例则是示范如何在运行时期设置 TMaskEdit 对象的掩码字符。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TMaskEdit 对象。
3. 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if not MaskEdit1.IsMasked then begin
    //1 代表要将掩码字符当作数据的一部分
    //0 则不代表掩码字符为数据的一部分
    MaskEdit1.EditMask := '!99/99/00;1;_';
    //设置 Text 属性
    MaskEdit1.Text := '  /  /  ';
  end;
end;
```

4. 保存工程。
5. 按【F9】键运行工程。工程运行结果与范例 7-9 完全一样，您必须输入指定的日期格式。

7.2.2　TMaskEdit 对象掩码字符

在 Delphi 6 中除了现成的掩码格式可以使用外，另允许用户自定义掩码格式。掩码格式可以分为“分栏符”、“空格符”、“是否将掩码字符保存为数据的一部分”及“掩码本身”等

四个部分来讨论。

- **分栏符(;)**

在图 7.13 中“;”符号代表掩码中的分栏符。

- **空格符(_)**

在图 7.13 中“_”符号代表掩码中的空格符。

- **掩码是否为数据的一部分**

在图 7.13 中“1”代表掩码字符会保存在数据中，如果是“0”，则代表不会将掩码字符保存在数据中。如果您想运行时期更改它，可以通过“MaskNoSave”属性值来改变它，详细语法请参考范例 7-10。

- **掩码本身**

掩码本身会使用一些特殊字符来代表特定的掩码格式(请参考表 7.10)。如果您想运行时期更改它，可以通过“EditMask”属性值来改变它，详细语法请参考范例 7-10。

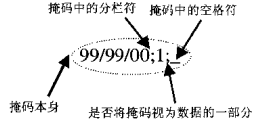


图 7.13 TMaskEdit 对象的掩码格式

表 7.11 TMaskEdit 对象的掩码格式字符

掩码字符	代表的意义
!	假如!字符出现在 TMaskEdit 掩码字符中，则“!”字符前的空格符不会被存为数据。反之，“!”字符后的空格符不会被存为数据
><	在掩码字符中，如果有“>”字符，则在“>”字符之后的字符皆会变成大写，一直到遇到“<”或字符串结束为止
<>	在掩码字符中，如果有“<”字符，则在“<”字符之后的字符都会变成小写，一直到遇到“>”或字符串结束为止
\	在掩码字符中，如果有“\”字符，则在“\”字符之后的特殊字符会被当成一般的文本处理
L	在掩码字符中，如果有“L”字符，则在其所对应的位置上只能输入大写或小写的英文字母，即为 A~Z 和 a~z
l	在掩码字符中，如果有“l”字符，则在其所对应的位置上只能输入大写或小写的英文字母，即为 A~Z 和 a~z。但不一定要输入

续上表

掩码字符	代表的意义
A	在掩码字符中，如果有“A”字符，则在其所对应的位置上只能输入英文字母或数字字符，即为 A~Z、a~z 或 0~9
a	在掩码字符中，如果有“a”字符，则在其所对应的位置上只能输入英文字母或数字字符，即为 A~Z、a~z 或 0~9，但不一定要输入
C	在掩码字符中，如果有“C”字符，则在其所对应的位置上只能输入字符
c	在掩码字符中，如果有“c”字符，则在其所对应的位置上只能输入字符，但不一定要输入
0	在掩码字符中，如果有“0”字符，则在其所对应的位置上只能输入数字
9	在掩码字符中，如果有“0”字符，则在其所对应的位置上只能输入数字，但不一定要输入
#	在掩码字符中，如果有“#”字符，则在其所对应的位置上只能输入数字或正负(加减)符号，但不一定要输入
:	在掩码字符中，如果有“:”字符，通常是用来分隔时间中的时分秒等数据
/	在掩码字符中，如果有“/”字符，通常是用来分隔日期中的年月日等数据
;	在掩码字符中的“;”字符，通常是用来分隔框中的三个部分
_	在掩码字符中的“_”字符会自动插入一个空格

7.3　*TMemo* 对象

您可以假想 TMemo 对象是由多个 TEdit 对象所组合的，它可以编辑多行的文本。TMemo 对象位于组件面板的 Standard 页标签中。表 7.12 为 TMemo 对象有关的属性，表 7.13 为操作方法。(注：加下划线的属性代表本章节会有详细说明的属性)

表 7.12

属性(Properties)	▲表示运行时期才能使用	
Alignment	▲CaretPos	Lines
ScrollBars	WantReturns	WantTabs
WordWrap	▲CanUndo	BorderStyle
HideSelection	MaxLength	Modified
OEMConvert	ReadOnly	SelLength
SelStart	SelText	▲Brush
▲ClientOrigin	▲ClientRect	▲ControlCount
▲Controls	Ctl3D	▲Handle
HelpContext	ImeMode	ImeName
ParentCtl3D	ParentWindow	▲Showing
TabOrder	TabStop	Align
Anchors	BiDiMode	BoundsRect
ClientHeight	ClientWidth	Color

续上表

属性(Properties)		▲表示运行时期才能使用
Constraints	ControlState	ControlStyle
Cursor	DragCursor	DragKind
DragMode	Enabled	Font
Height	Hint	Left
Name	Parent	ParentBiDiMode
ParentColor	ParentFont	ParentShowHint
PopupMenu	ShowHint	Text
Top	Visible	Width
WindowProc		

表 7.13

操作方法(Methods)		
Create	Destroy	Clear
ClearSelection	ClearUndo	CopyToClipboard
CutToClipboard	GetSelTextBuf	PasteFromClipboard
SelectAll	SetSelTextBuf	Undo
Broadcast	CanFocus	ContainsControl
ControlAtPos	CreateParented	DisableAlign
EnableAlign	Focused	GetTabOrderList
HandleAllocated	HandleNeeded	InsertControl
Invalidate	PaintTo	Realign
RemoveControl	Repaint	ScaleBy
ScrollBy	SetBounds	SetFocus
Update	UpdateControlState	BeginDrag
BringToFront	ClientToScreen	DragDrop
Dragging	EndDrag	GetTextBuf
GetTextLen	Hide	Perform
Refresh	ScreenToClient	SendToBack
SetTextBuf	Show	

由于 TMemo 对象大部分的属性值及操作方法都与 TEdit 对象相同，您可参考本章范例 7-1，不再赘述，以下仅对 TMemo 对象特有的属性加以说明。

Text 属性

Text 属性是获取或设置 TMemo 对象的全部内容。

Lines 属性

Lines 属性是获取或设置 TMemot 对象的单行数据内容。Lines 属性本身就是一个 TString 对象，它拥有自己的属性(Properties)及操作方法(Methods)例如添加(Add)、删除(Delete)及插入(Insert)等(详细数据请参阅第 15.2 节)。

范例 7-11

目的：本范例主要是说明 TMemo 对象的 Text 属性及 Lines 属性，因为 Lines 本身是一个 TString 对象，因此详细的用法请参阅第 15.2 节，本范例只示范附加(Addpend)、打开文件(LoadFromFile)及保存文件(SaveToFile)等。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 OpenDialog 对话框、一个 SaveDialog 对话框、一个 TMemo 对象、一个 TEdit 对象及三个 TButton 对象，对象摆设位置请参见图 7.14，其属性设置如表 7.14 所示。

表 7.14

对象　(Name 属性)	属性名称	属性设置值
OpenDialog	Name	OpenDialog1
SaveDialog	Name	SaveDialog1
TMemo	Name	Memo1
TEdit	Name	Edit1
TButton(Button1)	Caption	取出数据
TButton(Button2)	Caption	写入数据
TButton(Button3)	Caption	打开文件
TButton(Button4)	Caption	保存文件

步骤 3 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   //将 TMemo 对象的第一行取出来
   Edit1.Text:=Memo1.Lines[0];
end;
```

步骤 4 在 Button2 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   //将 TEdit 对象的内容加到 TMemo 对象的最后
   Memo1.Lines.Append(Edit1.Text);
end;
```

(5) 在 Button3 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   //打开文件
   if OpenDialog1.Execute then
     Memo1.Lines.LoadFromFile(OpenDialog1.fileName);
end;
```

(6) 在 Button4 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  //保存文件
  if SaveDialog1.Execute then
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

(7) 保存工程。

(8) 按【F9】功能键运行工程。工程运行后您可以利用“打开文件”、“保存文件”等功能以文件的方式处理 Memo1 对象的数据，您也可以通过 Lines 属性从 Memo1 对象获取或加入以行为单位的数据，如图 7.14 所示。

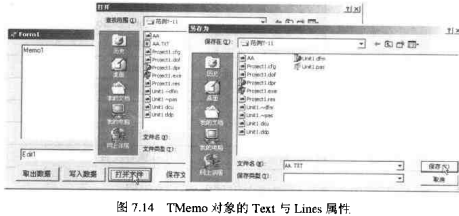


图 7.14 TMemo 对象的 Text 与 Lines 属性

ScrollBars 属性

ScrollBars 属性是用来决定 TMemo 对象要加入滚动条，ScrollBars 有水平滚动条(ssHorizontal)、垂直滚动条(ssVertical)、两者都有(ssBoth)或两者都没有(ssNone)。您可以在设置时期决定 ScrollBars 属性值，也可以在设计时期决定它。

范例 7-12

目的：本范例主要是说明 TMemo 对象的 ScrollBars 属性，您可以在设置时期设置，也可以通过运行时期设置 ScrollBars 属性值。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TMemo 对象及四个 TButton 对象，其对象摆设位置请参见图 7.15，其属性设置值如表 7-15 所示。

表 7-15

对象　(Name 属性)	属性名称	属性设置值
TMemo	Name	Memo1
TButton(Button1)	Caption	水平滚动条
TButton(Button2)	Caption	垂直滚动条
TButton(Button3)	Caption	两者都有
TButton(Button4)	Caption	两者都无

STEP 3 在 Button1 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //水平滚动条
  Memo1.ScrollBars:=ssHorizontal;
end;
```

STEP 4 在 Button2 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //垂直滚动条
  Memo1.ScrollBars:=ssVertical;
end;
```

STEP 5 在 Button3 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  //同有垂直及水平的滚动条
  Memo1.ScrollBars:=ssBoth;
end;
```

STEP 6 在 Button4 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  //无滚动条
  Memo1.ScrollBars:=ssNone;
end;
```

STEP 7 保存工程。

STEP 8 按【F9】功能键运行工程。工程运行后，您可以通过四个不同的 TButton 对象来动态决定 TMemo 对象的 ScrollBars 属性值，如图 7.15 所示。

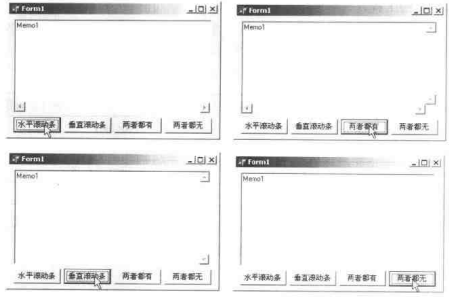


图 7.15 TMemo 对象的四种 ScrollBars 属性

WordWrap 属性

WordWrap 属性是用来决定 TMemo 对象中，当输入文本比 TMemo 对象的宽度还要长时的处理方式，当 WordWrap 属性值为 True，表示会将超出 TMemo 对象宽度的文本转至下一行；当 WordWrap 属性值为 False，则表示文本会无限往右延伸。WordWrap 属性值必须在 ScrollBars 属性值为 ssNone、ssVertical 时才会有效。

WantReturns 属性

WantReturns 属性是用来决定当用户按下【Enter】键时所作的处理方式，当 WantReturns 属性值为 True 时，则表示当用户按下【Enter】键时，会将输入点转至下一行；当 WantReturns 属性值为 False 时，则表示用户所按下的【Enter】键是要传给 TForm 对象的下一个焦点对象。如果您坚持要将 WantReturns 设置成 False，而且又想在 TMemo 对象中进行换行的动作(将输入点转至下一行)的话，那您可以同时按下【Ctrl+Enter】键来替换【Enter】键。

WantTabs 属性

WantTabs 属性与 WantReturns 属性相当类似，最大的不同是 WantTabs 属性的对象是【Tab】键，而 WantReturns 属性的对象是【Enter】键。WantTabs 属性是用来决定当用户按下【Tab】键时所作的处理方式，当 WantTabs 属性值为 True 时，则表示当用户按下【Tab】键时，会将输入点跳至下一个定位点；当 WantTabs 属性值为 False 时，则表示用户所按下的【Tab】键是要传给 TForm 对象的下一个焦点对象。

范例 7-13

目的：本范例主要是说明 TMemo 对象的 WordWrap 属性及【Enter】键与【Tab】键对 TMemo 对象的影响。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入一个 TMemo 对象，其属性设置如表 7.16 所示。

表 7.16

对象 (Name 属性)	属性名称	属性设置值
TMemo	Name	Memo1
	WordWrap	True
	WantEnters	False
	WantTabs	True
TButton	Name	Button1
	Default	True

STEP 3 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;
```

STEP 4 保存工程。

STEP 5 按【F9】键运行工程。由于我们将 WordWrap 属性值设为 True，所以当您输入太长的字符串时，系统会自动将输入点移至下一行；当您按下【Tab】键时，系统会自动将输入点跳到下一个定位点，那是因为我们将 WantTabs 属性设置为 True 的缘故。当您按下【Enter】键时，系统会调用默认的按钮(Button1)，然后触发 Button1 的 onClick 事件并结束系统，那是因为我们将 Button1 设置成系统默认的按钮，且在其 Click 事件处理过程中加入 Close 代码的缘故。

7.4　TRichEdit 对象

您可以将 TRichEdit 对象想象为 TMemo 对象的豪华版，它是支持 RTF 格式的 TMemo 对象，TRichEdit 可以编辑多行的文本，而其容量基本上是没有限制的。它最大的特点就是可以对对象中的文本分别改变字体、格式及拖动等简单文字处理功能。TRichEditt 对象位于组件面板的 Win32 页标签中。表 7.17 和表 7.18 将 TrichEdit 对象有关的属性及操作方法例表说明。(注：加底线的属性代表本章节会有详细说明的属性)

表 7.17

属性(Properties)	▲表示运行时期才能使用	
DefAttributes	DefaultConverter	GetCaretPos
HideScrollBars	HideSelection	Lines
PageRect	▲Paragraph	PlainText
SelAttributes	SelLength	SelStart

续上表

属性(Properties)	▲表示运行时期才能使用	
SelText	Alignment	▲CaretPos
ScrollBars	WantReturns	WantTabs
WordWrap	BorderStyle	▲CanUndo
MaxLength	Modified	ReadOnly
BorderWidth	▲Brush	▲ClientOrigin
▲ClientRect	▲ControlCount	▲Controls
Ctl3D	▲Handle	HelpContext
ImeMode	ImeName	ParentCtl3D
ParentWindow	▲Showing	TabOrder
TabStop	Align	Anchors
BiDiMode	BoundsRect	ClientHeight
ClientWidth	Color	Constraints
ControlState	ControlStyle	Cursor
DragCursor	DragKind	DragMode
Enabled	Font	Height
Hint	Left	Name
Parent	ParentBiDiMode	ParentColor
ParentFont	ParentShowHint	PopupMenu
ShowHint	Text	Top
Visible	Width	WindowProc

表 7.18

操作方法(Methods)		
Clear	Create	Destroy
FindText	GetSelTextBuf	Print
ClearSelection	ClearUndo	CopyToClipboard
CutToClipboard	PasteFromClipboard	Undo
SelectAll	SetSelTextBuf	Broadcast
CanFocus	ContainsControl	ControlAtPos
CreateParented	DisableAlign	EnableAlign
Focused	GetTabOrderList	HandleAllocated
HandleNeeded	InsertControl	Invalidate
PaintTo	Realign	RemoveControl
Repaint	ScaleBy	ScrollBy
SetBounds	SetFocus	Update
UpdateControlState	BeginDrag	BringToFront

续上表

操作方法(Methods)		
ClientToScreen	DragDrop	Dragging
EndDrag	GetTextBuf	GetTextLen
Hide	Perform	Refresh
ScreenToClient	SendToBack	SetTextBuf
Show		

这些属性及操作方法事实上大部分您都已经认识了，例如剪贴板的应用、字符串的选取等属性大都与 TEdit、TMemo 对象相同，因此相同的部分这里就不再里重复。以下分为两小节分别对 TRichEdit 对象的属性及操作方法加以说明。

7.4.1　TRichEdit 对象属性

由于 TRichEdit 对象支持 RTF 格式的 TMemo 对象，因此它必然有一些较为独特的属性，以下将较常用的属性加以说明。

DefAttributes 属性

DefAttributes 属性是用来设置 TRichEdit 对象的默认字体的属性，例如颜色、大小、字体等。

SelAttributes 属性

SelAttributes 属性是用来设置 TRichEdit 对象中被选取字符串的字体属性，例如颜色、大小、字体等。

范例 7-14

目的：本范例主要是说明 TRichEdit 对象的 DefAttributes 属性及 SelAttributes 属性的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TRichEdit 对象及两个 TButton 对象，其属性设置如表 7.19 所示。

表 7.19

对象　(Name 属性)	属性名称	属性设置值
TRichEdit(RichEdit1)	HideSelection	False
TButton(Button1)	Caption	字体默认值
TButton(Button2)	Caption	自定义字体

STEP 3 在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  RichEdit1.DefAttributes.Height:= 15;  {字符高度}
  RichEdit1.DefAttributes.Color := clBlue;  {颜色}
  RichEdit1.DefAttributes.Style := [fsBold, fsItalic];
end;
```

步骤 4 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  with RichEdit1.SelAttributes do
  begin
    Color := clRed;  {颜色}
    Size:=Size+2;    {大小}
    Name:='Arial';   {字体}
    {字体:粗体、斜体、下划线、删除线}
    Style:=[fsBold,fsItalic,fsUnderline,fsStrikeOut];
  end;
end;
```

步骤 5 保存工程。

步骤 6 按【F9】功能键运行工程，工程运行之后您可以利用“字体默认值”按钮来设置整个 TRichEdit 对象的字体默认值，您也可以先选择部分字符串，再单击“自定义字体”按钮来改变所选取字符串的字体设置值，如图 7.16 所示。



图 7.16　TRichEdit 对象的字体属性

- *HideScrollBars* 属性

HideScrollBars 属性是用来设置 TRichEdit 对象中的滚动条出现的时机，当 HideScrollBars 属性值为 True 时，表示当文本内容很多时才出现滚动条，否则就将滚动条加以隐藏；当 HideScrollBars 属性值为 False 时，表示不隐藏滚动条。HideScrollBars 属性必须在 ScrollBars 属性值不为 ssNone 才会有效。

- *PageRect* 属性

PageRect 属性是用来设置 TRichEdit 对象列表时的纸张大小。

Paragraph 属性

Paragraph 属性是用来设置 TRichEdit 对象指定段落的各项设置。Paragraph 属性继承自 TParaAttributes 对象。如果您想更进一步地了解它可以参考帮助说明。

PlainText 属性

PlainText 属性是用来决定打开某个.RTF 文件时，是否要显示.RTF 的格式，或是以一般的文本格式加以显示。当 PlainText 属性值为 False 时，表示以.RTF 格式显示，当 PlainText 属性值为 True 时，则是以一般文本格式显示。

范例 7-15

目的：本范例主要是说明 TRichEdit 对象的 Plain 属性的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入两个 TRichEdit 对象、一个 OpenDialog 对话框及一个 TButton 对象，其属性分别设置如表 7.20 所示。

表 7.20

对象　(Name 属性)	属性名称	属性设置值
TRichEdit(RichEdit1)	PlainText	True
TRichEdit(RichEdit2)	PlainText	False
TButton(Button1)	Caption	打开文件
TOpenDialog	Name	OpenDialog1

STEP 3 在 TButton 对象的 onClock 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with OpenDialog1 do
  if Execute then
     begin
     RichEdit1.Lines.LoadFromFile(FileName);
     RichEdit2.Lines.LoadFromFile(FileName);
     end;
end;
```

STEP 4 保存工程。

STEP 5 按【F9】功能键运行工程，工程运行后，您同时在两个 TRichEdit 对象中打开一个有.RTF 格式的文件(\Borland\Delphi6 \Demos\Richedit\overview.rtf)，此时分别以一般文本格式及 RTF 格式显示，如图 7.17 所示。

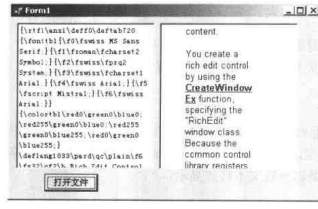


图 7.17 TRichEdit 对象的 PlainText 属性

7.4.2 TRichEdit 对象操作方法

以下将 TRichEdit 对象较为常用的操作方法加以说明。

- *FindText* 操作方法

FindText 操作方法是用来查找在 TRichEdit 对象中的某特定字符串。如果找到指定字符串则返回一个非－1 的整数。FindText 操作方法的 Options 是一个选择性的参数，其包含 stWholeWord 与 stMatchCase 两种，前者代表“全字匹配”，后者代表“区分大小写”或“大小写均符合”。其原型声明如下：

```
type
  TSearchType = (stWholeWord, stMatchCase);
  TSearchTypes = set of TsearchType;
Function FindText(const SearchStr: string; StartPos, Length
    Integer; Options: TSearchTypes): Integer;
```

- *Print* 操作方法

Print 操作方法是用来打印 TRichEdit 对象中的内容。

范例 7-16

目的：本范例主要是说明如何使用 TRichEdit 对象的 FindText 及 Print 等操作方法。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TRichEdit、两个 TButton 对象及一个 TFindDialog，其属性设置如表 7.21 所示。

表 7.21

对象 (Name 属性)	属性名称	属性设置值
TRichEdit(RichEdit1)	HideSelection	False

续上表

对象　(Name 属性)	属性名称	属性设置值
TButton(Button1)	Caption	查找
TButton(Button2)	Caption	打印
TFindDialog	Name	FindDialog1
	Options	[frDown,frMatchCase]

STEP 3 在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   //打开查找对话框
   FindDialog1.Execute;
end;
```

STEP 4 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //此文件名称会出现在打印机的任务栏中
  RichEdit1.Print('文件名称');
end;
```

STEP 5 在 TFindDialog 对象的 onFind 事件处理过程中加入以下的代码：

```
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  //用来判断是否找到了指定的字符串
  FoundFlag: LongInt;
  //用来记录所要查找字符串的起始位置
  StartPos:integer;
  //用来记录被查找的字符串范围长度
  StrLength:integer;
begin
  with RichEdit1 do
  begin
    //假如 SelLength<>0 表示在这之前已经找到一个字符串了
    //则在后面继续查找,否则从头开始查找
    if SelLength <> 0 then StartPos := SelStart + SelLength
    else  StartPos := 0;

    //设置被查找字符串范围长度
    StrLength := Length(Text) - StartPos;

    //开始查找
```

```
    FoundFlag := FindText(FindDialog1.FindText, StartPos,
                          StrLength, [stMatchCase]);
    if FoundFlag <> -1 then
      begin
       SetFocus;
       SelStart := FoundFlag;
       SelLength := Length(FindDialog1.FindText);
      end
    else
      //判断为找不到或已查找完毕
      if selLength<>0 then
        showmessage('文件已到结尾')
      else
        showMessage('找不到指定字符串');
  end;
end;
```

6 保存工程。

7 按【F9】运行工程，此时您可以利用“查找”按钮打开“查找”对话框以查找出您所指定的字符串，如图 7.18 所示。您也可以单击“打印”按钮打印目前 TRichEdit 对象的内容，如图 7.19 所示。

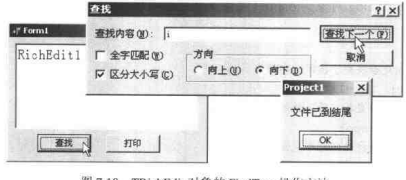


图 7.18 TRichEdit 对象的 FindText 操作方法

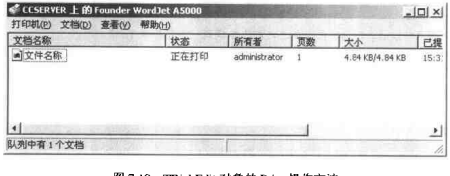


图 7.19 TRichEdit 对象的 Print 操作方法

7.5　TLabel 对象

在应用程序的发展中，我们常需要引导或提示信息给用户，而这种提示信息是不让用户变更的，虽然您可以使用 TEdit 对象将其 ReadOnly 属性设为 True 来完成。但事实上您有更好的选择，那就是 TLabel 对象。在前面六个章节的范例中，我们有很多使用这个对象的经验。表 7.22 和表 7.23 将 TLabel 对象的所有属性及操作方法加以列表说明（注：加下划线的属性代表本章节会有详细说明的属性）。

表 7.22

属性(Properties)		▲表示运行时期才能使用
Alignment	AutoSize	FocusControl
Layout	ShowAccelChar	Transparent
WordWrap	▲Canvas	Action
Align	Anchors	BiDiMode
BoundsRect	Caption	ClientHeight
▲ClientOrigin	▲ClientRect	ClientWidth
Color	Constraints	▲ControlState
ControlStyle	Cursor	DockOrientation
DragCursor	▲DragKind	▲DragMode
▲Enabled	▲Floating	▲FloatingDockSiteClass
Font	▲Height	Hint
HostDockSite	Left	LRDockWidth
Name	▲Parent	ParentBiDiMode
ParentColor	ParentFont	ParentShowHint
PopupMenu	ShowHint	TBDockHeight
Top	UndockHeight	UndockWidth
Visible	Width	WindowProc

表 7.23

操作方法(Methods)		
Create	Destroy	BeginDrag
BringToFront	ClientToScreen	Dock
DragDrop	Dragging	DrawTextBiDiModeFlags
EndDrag	GetControlsAlignment	DrawTextBiDiModeFlagsReadingOnly
GetTextBuf	GetTextLen	Hide
InitiateAction	Invalidate	IsRightToLeft
ManualDock	ManualFloat	Perform
Refresh	Repaint	ReplaceDockedControl

续上表

操作方法(Methods)		
ScreenToClient	SendToBack	SetBounds
SetTextBuf	Show	Update
UseRightToLeftAlignment	UseRightToLeftReading	UseRightToLeftScrollBar

以下针对 TLabel 对象一些常用的属性加以说明。

Caption 属性

Caption 属性是 Tlabel 对象所要显示的文本内容，您可以在 Caption 属性中某个字符左边加入“&”，则该字符即成了“快捷键”。在运行时期只要您同时按住【Alt】键与该字符即可启动“快捷键”。

FocuseControl 属性

FocusControl 属性是用来让 TLabel 对象与 TForm 中的其他对象进行连接。FocusControl 属性要有效，您必须在 TLabel 对象的 Capiton 属性设有快捷键。

范例 7-17

目的：本范例主要是说明如何使用 TLabel 对象的 Caption 与 FocusControl 等属性。

1. 创建新的工程。
2. 在 TForm 对象上加入两个 TLabel 对象及两个 TEdit 对象，其属性设置如表 7.24 所示。

表 7.24

对象(Name 属性)	属性名称	属性设置值
TLabel(Label1)	Capiton	&Name
	FocusControl	Edit1
TLabel(Label2)	Caption	PagerNum&ber
	FocusControl	Edit2
TEdit(Edit1)		
TEdit(Edit2)		

3. 保存工程。
4. 按【F9】键运行工程，工程运行后，您可以利用【Alt+N】键与【Alt+B】键使焦点在 Edit1 及 Edit2 之间互换，如图 7.20 所示。

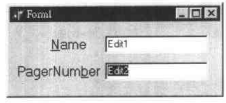


图 7.20　TLabel 对象的 FocusControl 属性

AutoSize 属性

AutoSize 属性决定 TLable 对象的大小是否要由文本内容来决定。当 AutoSize 属性值为 True，表示会由文本内容来决定 TLabel 对象的大小。

Alignment 属性

Alignment 属性是 TLabel 对象所要显示的文本内容的对齐方式，有左对齐(taLeftJustify)、右对齐(taRightJustify)及居中对齐(taCenter)，这些设置必须在 AutoSize 属性值为 False 情况下有效，如图 7.21 所示。



图 7.21　TLabel 对象的 Alignment 属性

Layout 属性

Layout 属性是 TLabel 对象所要显示文本内容的排列方式，向上对齐排列(tlTop)、向下对齐排列(tlCenter)及居中对齐排列(tlBottom)，这些设置必须在 AutoSize 属性值为 False 情况下有效，如图 7.22 所示。



图 7.22　TLabel 对象的 Layout 属性

Transparent 属性

Transparent 属性主要是决定 TLabel 对象是否为透明对象，当 Transparnet 属性值为 True，则表示 TLabel 对象为透明对象。反之，则为不透明对象，如图 7.23 所示。

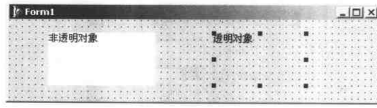


图 7.23　TLabel 对象的 Transparent 属性

WordWrap 属性

WordWrap 属性是决定 TLabel 对象是否具有多行显示的能力，当 WordWrap 属性值为 True 时，表示 TLabel 对象具有多行的显示能力。反之，则只有单行的显示能力。您无法直接在 Caption 输入多行的文本，但您可以在运行时期设置 Caption 属性。

范例 7-18

目的：本范例主要说明如何使用 TLabel 对象的 WordWrap 属性。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象上加入一个 TLabel 对象、一个 TMemot 对象及一个 TButton 对象，其属性设置如表 7.25 所示。

表 7.25

对象　(Name 属性)	属性名称	属性设置值
TLabel(Label1)	WordWrap	True
	Autosize	True(与运行结果无关)
TMemo	Name	Memo1
TButton	Name	Button1

步骤 3　在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption:=Memo1.Text;
end;
```

步骤 4　保存工程。

步骤 5　按【F9】键运行工程，工程运行后，您可以先再 TMemo 对象加入多行的文本，再利用 Button1 将 Memo1 对象的内容复制到 TLabel 中，如图 7.24 所示。

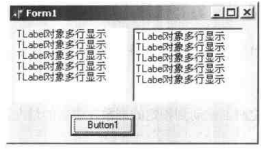


图 7.24　TLabel 对象的 WordWrap 属性

7.6　TStringGrid 对象

本节要介绍 TStringGrid 对象，此对象位于 Additional 的组件面板上，其常用的功能是显示数量多且固定的数据上，图 7.25 为 TStringGrid 对象的外观。

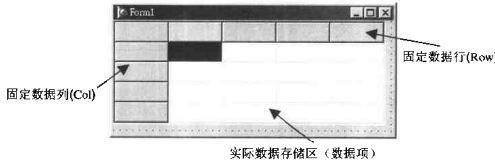


图 7.25　TStringGrid 的外观说明

以下就 TStringGrid 对象经常被使用的属性加以说明。

- *ColCount* 属性

属性 Colcount 决定 Grid 的列数，包括固定列数与实际访问的数据列数，初值为 5。

- *RowCount* 属性

属性 RowCount 决定 Grid 的行数，包括固定行数与实际存取的数据行数，初值为 5。

- *FixedCols* 属性

属性 FixedCols 决定 Grid 的固定列数。

- *FixedRows* 属性

属性 FixedRows 决定 Grid 的固定行数。

- *DefaultColWidth* 属性

属性 DefaultColWidth 决定每一列的宽度。

DefaultRowHeight 属性

属性 DefaultRowHeight 决定每一行的高度。

LeftCol 属性

属性 LeftCol 可以指定画面滚动到指定的列数在画面的最左侧。只能在运行时使用，方法如下：

```
//代表将 Grid 滚动到第五列的位置，其列行索引值由 0 开始计算。
LeftCol := 4;
```

TopRow 属性

属性 TopRow 可以指定画面滚动到指定的行数在画面的最上方。只能在运行时使用，方法如下：

```
//代表将 Grid 滚动到第五行的位置，其列行索引值由 0 开始计算。
TopRow := 4;
```

Color 属性

属性 Color 决定整个 Grid 数据区的颜色。

FixedColor 属性

属性 FixedColor 决定固定行列的颜色。

Options 属性

属性 Options 的数据类型是一个集合数据类型，Options 属性用来决定 Grid 的外观及相关作用。用表 7.26 方式说明 Options 的可能值与其所代表的意义。

表 7.26

可能值	说明
goFixedHorzLine	在数据项和固定行之间显示水平线
goFixedVertLine	在数据项和固定列之间显示垂直线
goHorzLine	在数据项之间显示水平线
goVertLine	在数据项之间显示垂直线
goDrawFocusSelected	当用户同时选取多个数据项时，第一数据项会反白
goEditing	允许用户直接在数据项上输入数据，但在此模式中用户无法使用鼠标同时选取多个数据项
goAlwaysShowEditor	此选项必须和 goEditing 一起使用。若 goAlwayShowEditor 被设为 True 时，则用户使用鼠标选择某个数据项时，会自动显示插入点等待输入；若其值设为 False，则鼠标选择某个数据项时，必须要按下【Enter】键或是【F12】键才会产生插入点

续上表

可能值	说明
goTabs	允许用户使用【Tab】键和【Shift+Tab】键在数据项上移动光标
goRowSelect	用户在某个数据项上点选鼠标，可以选取一行，但不能与 goRangeSelect 同时设置
goRangeSelect	允许用户利用鼠标同时选取多个数据项
goRowSizing	允许用户利用鼠标改变数据行的高度
goColSizing	允许用户利用鼠标改变数据列的宽度
goRowMoving	允许用户利用鼠标移动数据行
goColMoving	允许用户利用鼠标移动数据列
gothumbTracking	此选项主要是用来设置滚动条的状态。若此选项被设为 True 时，当用户拖动滚动条时 Grid 对象也会一起滚动；反之，则必须在滚动条被拖动后 Grid 才会滚动

Cells 属性

属性 Cells 为填入数据所使用的属性，语法如下：

```
Property cells[ACol, ARow : Integer] : string;
```

数据写入是由列数及行数形成的坐标所决定的，其中两个参数必须为整数类型。

范例 7-19

目的：本范例主要是说明 TStingGrid 对象如何写入数据。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TStingGrid 对象、一个 TButton、三个 TLabel 对象、两个 TSpinEdit 对象及一个 TEdit 对象。其相关属性设置如表 7.27 所示。

表 7.27

对象	属性名称	属性设置值
TStringGrid	Name	StringGrid1
	ColCount	6
	RowCount	6
TButton	Name	Button1
	Caption	确定
TLabel	Name	Label1
	Caption	选择列号(Col)
TLabel	Name	Label2
	Caption	选择行号(Row)
TLabel	Name	Label3
	Caption	输入字符串

续上表

对象	属性名称	属性设置值
TSpinEdit	Name	SpinEditCol
TSpinEdit	Name	SpinEditRow
TEdit	Name	Edit1

步骤 3 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //设置 SpinEdit 的初值，因为 StringGrid 是从 0 开始算的，
  //因此要减 1
  SpinEditRow.MaxValue:= StringGrid1.RowCount-1;
  SpinEditRow.MinValue:= 0;
  SpinEditCol.MaxValue:= StringGrid1.ColCount-1;
  SpinEditCol.MinValue:= 0;
end;
```

步骤 4 在 Button1 的 OnClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //将用户输入的字符串，在 StringGrid 指定位置加以输出来
  StringGrid1.Cells[SpinEditCol.Value,SpinEditRow.Value]:=
                          Edit1.Text;
end;
```

步骤 5 保存工程，并按下【F9】功能键运行工程。工程运行后，您可以输入指定的字符串，并指定坐标将此字符串显示，如图 7.26 所示。



图 7.26 TStringGrid 对象的运行结果

7.7 TLabeledEdit 对象

TLabeledEdit 对象可以视为 TLabel 与 TEdit 对象的结合体，其位于组件面板上的

Additional 页标签上。由于 TLabeledEdit 对象有很多属性和事件与先前所介绍的 TLabel 与 TEdit 对象相同，因此笔者这里仅就其特别的属性加以介绍，其他的属性与事件请分别参考 TLabel 与 TEdit 对象相关章节的详细说明。

EditLabel 属性

EditLabel 属性继承自 TBoundLabel 对象类型，其大部分的属性与 TLabel 对象相同。在设计时期，您可以通过对象检查器修改 Label 的相关属性，如图 7.27 所示。

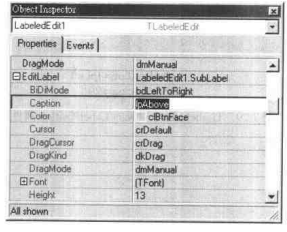


图 7.27　设计时期修改 TLabeledEdit 的 EditLabel 属性

如果您想运行时期修改 Label 对象的 Caption 属性，请参考下面的范例说明，这里以 LabeledEdit1 为例。

```
LabeledEdit1EditLabelCaption := '测试标题';
```

LabelPosition 属性

LabelPosition 属性是用来设置 Label 对象的显示位置，其包含 lpAbove、lpBelow、lpLeft 及 lpRight 等。LpAbove 代表 Label 对象将显示在 Edit 对象的上方；lpBelow 代表 Label 对象将显示在 Edit 对象的下方；lpLeft 代表 Label 对象将显示在 Edit 对象的左方；lpRight 代表 Label 对象将显示在 Edit 对象的右方，如图 7.28 所示。

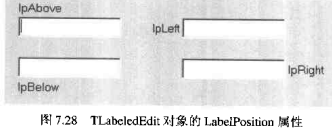


图 7.28　TLabeledEdit 对象的 LabelPosition 属性

LabelSpacing 属性

LabelSpacing 属性是用来设置 Label 对象与 Edit 对象间的距离，如图 7.29 所示。

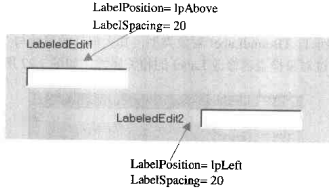


图 7.29　TLabeledEdit 对象的 LabelPosition 属性

第8章

Delphi 6 功能选择对象

本章目标

在前面几个章节中，我们已经介绍了 Delphi 6 常用的按钮对象与文字控制对象。这些都是基本且常用的对象，接着，本章将为大家介绍 Delphi 6 的功能选择对象，包括 TMainMenu、TPopupMenu、TListBox、TComboBox、TComboBoxEx、TRadioButton、TCheckBox、TCheckListBox、TTreeView 及 TListView 等对象。

本章主题

8.1 TmainMenu 对象

8.2 TPopupMenu 对象

8.3 TListBox 对象

8.4 TComboBox 对象

8.5 TComboBoxEx 对象

8.6 TRadioButton 对象

8.7 TCheckBox 对象

8.8 TCheckListBox 对象

8.9 TTreeView 对象

8.10 TListView 对象

8.1 TMainMenu 对象

在窗口程序设计中 TMainMenu 对象是一个非常重要的对象，它可以快速而便利地完成下拉式菜单的制作，其位于组件面板上的 Standard 页标签中。TMainMenu 对象可以协助您将一个庞大的系统功能做有规则的呈现。本小节将从下列几个角度来讨论 TMainMenu 对象。

8.1.1 下拉式菜单的制作

制作一个下拉式菜单对于 Delphi 6 而言是一件非常容易的事，您只要按照下列步骤即可完成它。

步骤 1 在 TForm 对象上加入一个 TMainMenu 对象，此时系统会自动为 TForm 对象的 Menu 属性填入此 TMainMenu 对象的名称。

步骤 2 在 TMainMenu 对象图标上双击或点选对象检查器中的 Items 属性，此时系统出现一个菜单编辑器。

步骤 3 请在对象检查器中的 Caption 属性输入指定的菜单名称，这里以“主选项”为例，如图 8.1 所示。

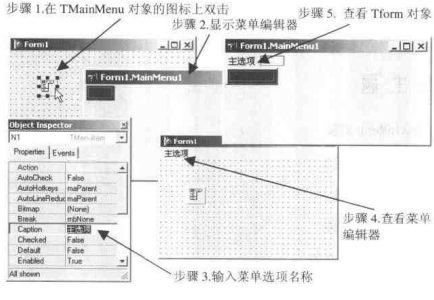


图 8.1 为 TMenuItem 对象创建子菜单

步骤 4 按照步骤 3 加入其他子选项，这里以“选项一”、“选项二”、“-”及“选项三”为例。当您输入“-”符号时，系统会视为“分隔符”。

步骤 5 查看 TForm 对象，此时您可以发现 TForm 对象上已经出现一个包含三个子选项的菜单，如图 8.2 所示。

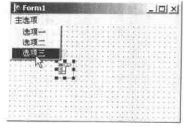


图 8.2　查看下拉式菜单



如果您想为某个选项加入子菜单(SubMenu)，您只要在 TMenuItem 对象上按右键，并选择 Create Submenu 功能即可轻松创建一个子菜单，如图 8.3 所示。

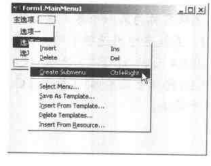


图 8.3　为 TMenuItem 对象创建子菜单

8.1.2　*TMainMenu* 对象属性

在本小节中，我们将介绍 TMainMenu 对象的常用属性，包括：Items、AutoHotkeys、AutoLineReduction、AutoMerge 及 Images 等。

Items 属性

Items 属性继承自 TMenuItem 对象，在菜单中的每一个选项都是一个 TMenuItem 对象，我们将在下一个小节中讨论它。

AutoHotkeys 属性

AutoHotkeys 属性用来决定是否自动产生 HotKey（热键）的功能，当 AutoHotkeys 属性值为 maAutomatic，表示自动产生 HotKey；反之，当 AutoHotkeys 属性值为 maManual，表示系统不会自动产生 HotKey，如图 8.4 所示。

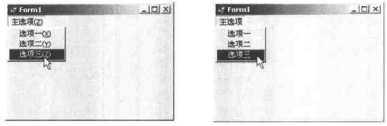


图 8.4 AutoHotkeys 属性

注意

HotKey 可提供用户使用键盘按键来操控 TMainMenu 对象。如果您将 AutoHotkeys 的属性值设为 maAutomatic，一般来说，系统自动命名的规则如下：当选项名称为英文时，系统会以选项名称的第一个英文字母为 HotKey，如遇重复则往后顺延。当选项名称为中文时，系统会根据选项个数，由字母 Z 往前排列。那用户又该如何操控主菜单呢?以图 8.4 而言，除了“主选项”使用【Alt】键+“英文字母键(【Z】)”显示外，其余的子选项(包括子菜单)都可使用【Ctrl】键+“英文字母键(【X】、【Y】、【Z】)”显示菜单。

AutoLineReduction 属性

AutoLineReduction 属性用来决定是否自动清除多余的“分隔符”功能，当 AutoLineReduction 属性值为 maAutomatic，表示自动清除；反之，当 AutoLineReduction 属性值为 maManual，表示系统不会自动清除，如图 8.5 所示。

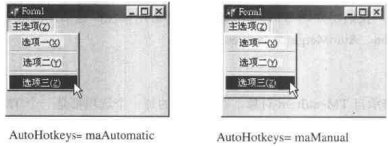


图 8.5 AutoLineReduction 属性

AutoMerge 属性

AutoMerge 属性用于决定是否将两个 TMainMenu 对象菜单加以合并，通常用于 MDI(Multiple Document Interface，多文档界面)应用程序。

Images 属性

Images 属性可连接至 TImageList 对象，通过 TImageList 对象可为每一个菜单选项显示小图标。如何为 TImageList 对象加载图片？请参考范例 6-4 的详细示范。如何为每一个菜单选项加入图标，请参考下一小节的详细说明。

8.1.3　TMenuItem 对象

在 TMenuMenu 对象菜单中的每一选项都是一个 TMenuItem 对象，在本小节中，我们将介绍 TMenuItem 对象的常用属性与操作方法，包括：Caption、Enabled、Action、ImageIndex、Bitmap、SubMenuImages、AutoHotkeys、AutoLineReduction、ShortCut、Break、GroupIndex、RadioItem、AutoCheck、Checked 等属性与 onClick 事件。

- *Caption 属性*

Caption 属性代表此 TMenuItem 对象所显示的文字。

- *Enabled 属性*

Enabled 属性决定此 TMenuItem 对象是否允许被使用。

- *Action 属性*

Action 属性通常连接到 TActionList 对象中的一个 Action Item。关于如何使用 Action Item，可参考第 13.1 节的 TActionList 对象介绍。

- *ImageIndex 属性*

ImageIndex 属性通常连接到 TImageList 对象中的一个图形文件，通常必须配合 TMainMenu 或 TPopupMenu 对象的 Images 属性一起使用。关于如何使用 TImageList 对象，请参考第 12.2 节的 TImageList 对象介绍。以下将示范如何为每一个 TMenuItem 对象加入图标。

1. 在 TForm 对象上加入一个 TMainMenu 对象并为其创建相关的菜单，如第 8.1.1 节所示。在 TForm 对象上加入一个 TImageList 对象并为其加入几个图形文件，请参考范例 6-4 的详细示范。
2. 在菜单编辑器中选择每一个 TMenuItem 对象，并通过对象检查器设置 ImageIndex 属性，如图 8.6 所示。

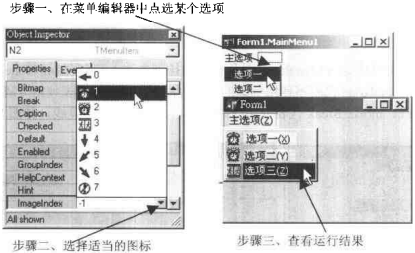


图 8.6　TMenuItem 对象的 ImageIndex 属性

Bitmap 属性

ImageIndex 属性是连接到 TImageList 对象中的一个图形文件，而 Bitmap 属性则是直接连接到一个图形文件，这两个属性都是为了让 TMenuItem 对象具有显示图片的能力。Bitmap 属性的设置方式与 TForm 对象的 Icon 属性相同，请参考第 5.1.2 节。

SubMenuImages 属性

SubMenuImages 属性通常连接到 TImageList 对象，以便子菜单(SubMenu)可以通过 ImageIndex 属性连接到 TImageList 对象中的一个图形文件。关于如何使用 TImageList 对象，请参考第 12.2 节的 TImageList 对象介绍。

AutoHotkeys 属性

AutoHotkeys 属性决定 TMenuItem 对象是否自动产生 HotKey。当 AutoHotkeys 属性值为 maAutomatic，则表示自动产生；AutoHotkeys 属性值为 maManual，则不自动产生；假如 AutoHotkeys 属性值为 maParent，表示视其父对象的 AutoHotkeys 属性值而定，其默认值为 maParent。

AutoLineReduction 属性

AutoLineReduction 属性用来决定是否自动清除多余的“分隔符”功能，当 AutoLineReduction 属性值为 maAutomatic，表示自动清除；当 AutoLineReduction 属性值为 maManual，表示系统不会自动清除；当 AutoLineReduction 属性值为 maParent，表示视其父对象的 AutoLineReduction 属性值而定，其默认值为 maParent。

ShortCut 属性

AutoHotkeys 属性是决定 TMenuItem 对象是否自动产生 HotKey，而 ShortCut 属性是允许用户自定义 HotKey。

Break 属性

Break 属性可以决定 TMenuItem 对象显示的排列方法，其包含 mbNone、mbBreak 及 mbBarBreak 等。mbNone 代表不使用分隔的显示方法；mbBreak 代表使用分隔的显示方法，但不使用分隔线；mbBarBreak 代表使用分隔的显示方法并使用分隔线，如图 8.7 所示。

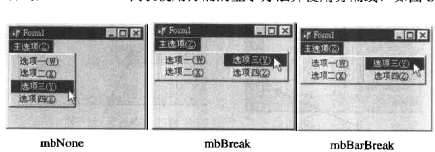


图 8.7 TMenuItem 对象的 Break 属性

GroupIndex 属性

GroupIndex 属性是代表每一个下拉式菜单的组合编号(默认值为 0)，例如图 8.7 的“主选项”。GroupIndex 属性可用于 MDI 应用程序的菜单合并，详细范例说明请参考 MDI 相关章节。

RadioItem 属性

RadioItem 属性代表 TMenuItem 对象是一个 RadioItem 对象，因此相同组合的 TMenuItem 对象同时只能有一个选项被选取。

AutoCheck 属性

AutoCheck 属性是代表当用户点选此 TMenuItem 对象时，系统会自动做检查的操作。当 AutoCheck 属性为 True 且 ReaioItem 属性为 False 时，代表此选项与 TCheckBox 对象具有相同的功能。有关 TCheckBox 对象的详细介绍请参考第 8.4 节的详细说明。当 AutoCheck 属性为 True 且 ReaioItem 属性为 True 时，代表此选项与 TRadioButton 对象具有相同的功能。有关 TRadioButton 对象的详细介绍请参考第 8.6 节的详细说明。

Checked 属性

Checked 属性可以用来决定此 TMenuItem 对象是否被选取，无论 RadioItem 属性为 True 或 False 都适用。图 8.8 中“选项一”是一个 RadionItem 对象，而且被选取；“选项二”同样被选取，但它不是一个 RaionItem 对象。

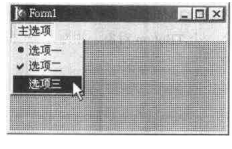


图 8.8　TMenuItem 对象的 Checked 属性

onClick 事件

TMenuItem 对象的 onClick 事件与其他对象的 onClick 事件用途相同，笔者不再赘述。这里仅示范如何为某个菜单创建 onClick 事件处理过程。您可以在 TForm 对象上点选某个菜单选项或在菜单编辑器上点选某个选项，如图 8.9 所示。

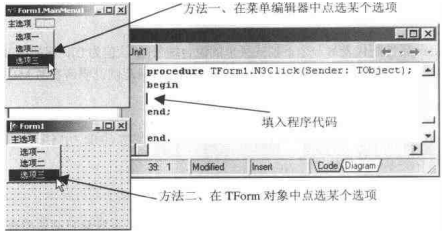


图 8.9 TMenuItem 对象的 onClick 事件处理

8.1.4 动态创建 TMainMenu 对象菜单

笔者将以范例的方式说明如何动态创建 TMainMenu 对象菜单。

范例 8-1

目的：本范例主要是说明如何动态创建 TMainMenu 对象菜单。

1. 创建新工程。
2. 在 TForm 对象上加入一个 TMainMenu 对象并创建四个选项，如图 8.10 所示。
3. 在 TForm 对象上加入一个 TButton 对象，并为其 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  NewItem: TMenuItem;
begin
  for i := 1 to 3 do begin
      //创建一个 MenuItem
      NewItem := TMenuItem.Create(MainMenu1);
      //为 TMainMenu 的第一个选项创建子菜单
      MainMenu1.Items[0].Items[0].Add(NewItem);
      //由父选项的 Caption 属性组合出此选项的名称
      NewItem.Caption := NewItem.Parent.Caption + '  ' +
IntToStr(I);
      //设置 onClick 事件
      NewItem.OnClick := myMenuItemsClick;
  end;
```

```
end;
```

步骤 4　创建一个新的过程以作为新菜单的 onClick 事件处理过程，请特别注意，因为这是使用自定义函数，因此必须在 TForm 对象的函数声明中加入此选项。

```
//函数声明
  public
    { Public declarations }
     procedure myMenuItemsClick(Sender: TObject);
  end;

//函数内容
procedure TForm1.myMenuitemsClick(Sender: TObject);
begin
  //显示该对象的 Caption 属性
  Showmessage((Sender as TMenuItem).Caption);
end;
```

步骤 5　运行工程并按下 Button1，此时系统自动为“选项一”选项加入三个子菜单，如图 8.10 所示。您同时也可以测试一下新菜单的 onClick 事件是否能成功的显示该菜单的 Caption 属性值。

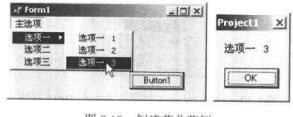


图 8.10　创建菜单范例

8.2　TPopupMenu 对象

在窗口程序设计中，TPopupMenu 对象也是一个常用的对象，它可以快速且便利的完成 Popup 菜单制作，其位于组件面板上的 Standard 页标签中。TPopupMenu 对象可以协助您在画面的任何一个地方显示适当的 Popup 菜单。通常一个 TForm 对象只会有一个 TMainMenu 对象，但却可以同时包含多个 TPopupMenu 对象。我们曾经在第 5 章的范例 5-10 中示范如何为 TPopupMenu 对象加入新的选项与事件处理，因此本小节不再赘述这些部分。以下仅就较常用的属性与操作方法加以说明。

Items 属性

Items 属性继承自 TMenuItem 对象，在菜单中的每一个选项都是一个 TMenuItem 对象，有关 TMenuItem 对象请参考第 8.1.3 节的详细说明。

Alignment 属性

Alignment 属性决定 TPopupmenu 对象出现的位置，其包含 paLeft、paRight 及 paCenter 等。paLeft 代表 TPopupmenu 对象出现在鼠标所在的左下方；paRight 代表 TPopupmenu 对象出现在鼠标所在的右下方；而 paCenter 则代表 TPopupmenu 对象出现在鼠标所在的中间位置，如图 8.11 所示。



paLeft　　paCenter　　paRight

图 8.11　TPopupmenu 对象的 Alignment 属性

AutoHotkeys 属性

AutoHotkeys 属性决定 TPopupmenu 对象是否自动产生 HotKey。当 AutoHotkeys 属性值为 maAutomatic，则表示自动产生；反之，则不自动产生，如图 8.12 所示。

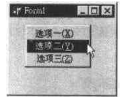


图 8.12　TPopupmenu 对象的 AutoHotkeys 属性

AutoPopup 属性

AutoPopup 属性决定 TPopupmenu 对象是否自动显示。当 AutoPopup 属性值为 True，表示自动显示；反之，则不自动显示。

Images 属性

Images 属性决定 TPopupmenu 对象所使用的图标来源，其通常是指一个 TImageList 对象。关于 TImageList 对象，请参考章节 12.1 的详细介绍。至于如何为每一个 ImenuItem 对象指定图标，请参考 8.1.3 章节的详细说明。

Popup 操作方法

Popup 操作方法可以让我们以手动的方式显示 TPopupmenu 对象。



8-2

目的：本范例将示范如何手动显示 TPopupmenu 对象。

1 创建新的工程。

(2) 在 TForm 对象上加入一个 TPopupmenu 对象与一个 TButton 对象。请自行设置 TPopupmenu 对象的显示选项，并在 TButton 对象的 onClick 事件处理过程中加入以下的代码。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  v_left : Integer;
  v_top  : Integer;
begin
  //设置显示坐标(您可以任意调整)
  v_left := left + Button1.left ;
  v_top  := top  + Button1.top  + (Button1.height * 2);
  //显示 TPopubmenu 对象
  Popupmenu1.Popup(v_left,v_top);
end;
```

(3) 保存工程，并按下【F9】功能键运行工程。工程运行后，当您按下 TButton 对象的同时 TPopupmenu 对象也一并显示，如图 8.13 所示。

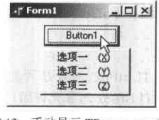


图 8.13　手动显示 TPopupmenu 对象

8.3　TListBox 对象

TListBox 对象可将所有可能的数据加以显示供用户选择，以节省用户输入数据的时间，其位于组件面板上的 Standard 页标签中。当 MultiSelect 属性值为 True 时，用户只要通过 Shift 键或 Ctrl 键就可以在 TListBox 对象上同时选取一个以上的数据项。以下就 TListBox 对象经常被使用的属性及操作方法加以说明。

8.3.1　TListBox 对象属性

Items 属性

Items 属性是真正保存 TListBox 对象数据项的属性，Items 属性是继承自 TString 对象，有关 TString 对象的详细操作方法请参考第 15.2 节。至于运行时期改变 Items 属性值常用的操作方法请参考第 8.3.2 节。如果您想在设计时期决定 Items 属性值，只要在 Items 属性值字段上用鼠标双击，通过“String List Editor”对话框进行编辑即可，如图 8.14 所示。

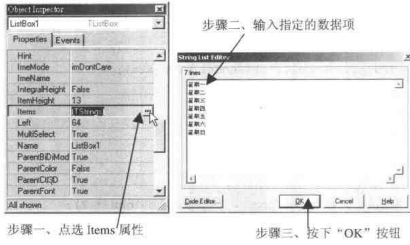


图 8.14　利用 String List Editor 编辑 TListBox 的数据项

Sorted 属性

Sorted 属性会决定此 TListBox 对象所显示数据项是否先经过排序的处理。当 Sorted 属性值为 True，表示数据会经过排序处理；反之，则以用户原先所输入的顺序为主。

MultiSelect 属性

MultiSelect 属性用来决定此 TListBox 对象是否允许用户同时选取多个数据项。当 MultiSelect 属性值为 True，表示此 TListBox 对象允许用户同时选取多个数据项，否则只能选取一个数据项。

SelCount 属性

SelCount 属性主要是记录用户在 TListBox 对象中的选取项目总数，通常 SelCount 属性必须在 MultiSelect 属性为 True 才会有效。

Selected 属性

Selected 属性用来判断某个数据项是否被选取。通常使用于当 MultiSelect 属性值为 True 的情况。

ItemIndex 属性

ItemIndex 属性是一个运行时期的属性，因此您无法通过对象检查器(Object Inspector)来设置其属性值，ItemIndex 属性代表目前 TListBox 对象中被选取的项目编号，通常使用于 MultiSelect 属性值为 False 时。假如 ItemIndex 属性值为 0，表示第一个选项被选取；反之，如果是-1，则表示没有任何选项是被选取的。

AutoComplete 属性

当 AutoComplete 属性为 True 时，用户在 TlistBox 对象上输入某个字(英文字母或中文字

都适用)时，系统会自动跳至第一个字符合条件的选项上。通常会配合 Sorted 属性为 True 一起使用。

范例 8-3

目的：本范例主要在说明 TListBox 对象的 MultiSelect、SelCount 及 Seleced 等各项属性的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TListBox 对象及一个 TButton 对象，其属性值设置如表 8.1 所示。

表 8.1

对象	属性名称	属性设置值
TListBox	Name	ListBox1
	MultiSelect	True
TButton	Name	Button1
	Caption	确定

STEP 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I,J:Integer;
  str:String;
begin
  Str:='您共选取'+IntToStr(ListBox1.SelCount)+
'个数据项' + #10;
  J:=0;
  for i := 0 to (ListBox1.Items.Count - 1) do
  if ListBox1.Selected[i] then
    begin
     J:=J+1;
     Str:=Str+'第'+IntToStr(J)+'个是数据项是:' +
ListBox1.Items[i] + #10;
    end;
  showMessage(Str);
end;
```

STEP 4 保存工程。

STEP 5 按【F9】功能键运行工程，工程运行后，您可以先选取某此数据项，再按下 TButton 对象确定您所选取的数据项，如图 8.15 所示。

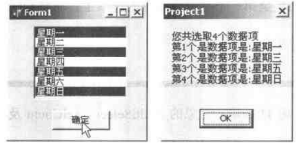


图 8.15 TListBox 对象的 MultiSelect 属性

Columns 属性

Columns 属性是决定此 TListBox 对象显示数据项的列数，默认值为 0，表示数据项以一个字段来显示。当 Columns 属性值大于 0，表示数据项以指定列数显示，如图 8.16 所示。

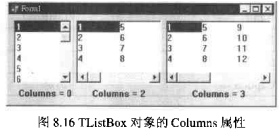


图 8.16 TListBox 对象的 Columns 属性

范例 8-4

目的：通过掷骰子范例说明如何使用 TListBox 对象的各项属性值。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TLabel 对象、一个 TButton 对象、一个 TImage 对象及一个 TListBox 对象。有关 TImage 对象的详细介绍，请参考第 12.1 节的详细说明。请将新加入的对象相关属性设置如表 8.2 所示。

表 8.2

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	掷骰子
TButton	Name	Button1
	Caption	随机产生点数
TImage	Name	Image1
	AutoSize	True
	Picture	1.bmp
TListBox	Name	ListBox1
	Items	一点、二点、三点、四点、五点、六点

STEP 3 在 TListBox 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
  MyFileName:String;
begin
  //用户指定点数
  MyFileName:=IntToStr(ListBox1.ItemIndex+1)+'.bmp';
  Image1.Picture.LoadFromFile(MyFileName);
end;
```

STEP 4 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyFileName:String;
begin
  //随机产生一个点数
  MyFileName:=IntToStr(Random(6)+1)+'.bmp';
  Image1.Picture.LoadFromFile(MyFileName);
end;
```

STEP 5 保存工程。

STEP 6 按下【F9】运行工程。工程运行后，您可以指定骰子的点数，或是由随机数产生点数。无论您是使用哪一种方式改变骰子的点数，系统会实时更新所对应的图片，如图 8.17 所示。

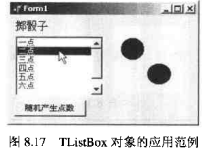


图 8.17　TListBox 对象的应用范例

注意　如果您运行上述范例很多次，您可能会发现为什么每一次系统所产生的随机数顺序都是一样的呢?那是因为系统每一次都使用相同的随机数表，所以运行结果当然相同。如果您想真正的产生随机数，也就是说，每一次的运行结果都不相同。您只要在调用 Random 函数之前，先调用 Randomize 函数即可。

8.3.2　TListBox 对象操作方法

在 TListBox 对象的操作方法中，最重要的要算是数据项(Items 属性)的处理。因此以下对有关 Items 属性的操作方法加以说明。

Clear 操作方法

Clear 操作方法可以清除 TListBox 对象上所有数据项。

Add 操作方法

Add 操作方法可为 TListBox 对象加入一个新的选项。

Insert 操作方法

Insert 操作方法可为 TListBox 对象在指定的索引位置(index)加入一个新的选项。

Delete 操作方法

Delete 操作方法可以删除 TListBox 对象指定索引位置(Index)的数据项。

Exchange 操作方法

Exchange 操作方法可以将 TListBox 对象中两个指定索引位置(Index)的数据项互换。

Move 操作方法

Move 操作方法可以将 TListBox 对象中两个指定索引位置(Index)的数据项互换。选项移动后，在 TListBox 对象中的数据项索引值会自动依照新的数据项顺序排列。



8-5

目的：本范例主要是说明 TListBox 对象常用的各种操作方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TListBox 对象、三个 TButton 对象。其相关属性设置如表 8.3 所示。

表 8.3

对象	属性名称	属性设置值
TListBox	Name	Listbox1
TButton	Name	ButtonAppend
	Caption	添加
TButton	Name	ButtonDelete
	Caption	删除
TButton	Name	ButtonClear
	Caption	清除
TButton	Name	ButtonModify
	Caption	修改

3 在 ButtonAppend 对象的 onClick 事件处理过程中加入以下的代码。在下列的代码中，笔者通过 InputQuery 对话框获取用户输入的新选项内容。有关 InputQuery 对话框请参考第 10.5 节的详细说明。

```
procedure TForm1.ButtonAppendClick(Sender: TObject);
var
  DefStr:String;
begin
  //输入指定数据
  defStr:='数据';
  if InputQuery('请输入 TListBox 的数据项','选项数据:',defStr)
    then
      ListBox1.items.add(defstr);
end;
```

4 在 ButtonDelete 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
begin
  //删除某个指定选项
  ListBox1.Items.Delete(ListBox1.Itemindex);
end;
```

5 在 ButtonClear 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonClearClick(Sender: TObject);
begin
  //清除所有选项
  ListBox1.Clear;
end;
```

6 在 ButtonModify 对象的 onClick 事件处理过程中加入以下的代码。在下列的代码中，笔者通过 InputQuery 对话框显示原有的选项内容，也允许用户输入新的项目内容。有关 InputQuery 对话框请参考第 10.5 节的详细说明。

```
procedure TForm1.ButtonModifyClick(Sender: TObject);
var
  DefStr:String;
begin
  //修改指定数据
  if (ListBox1.Items.Count > 0 ) and (ListBox1.ItemIndex >= 0 )
    then
    begin
      defStr:= ListBox1.Items[ListBox1.Itemindex];
      if InputQuery('请输入新的选项内容','选项新内容:',defStr)
        then
          ListBox1.items[ListBox1.ItemIndex] := defstr;
```

```
    end;
end;
```

7 保存工程。

8 按【F9】功能键运行工程。工程运行后，您可以任意的添加或是删除数据项，如图 8.18 所示。

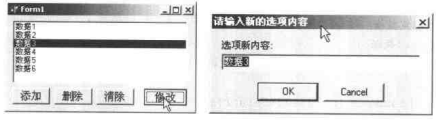


图 8.18 TListBox 对象的各种操作方法

8.4 TComboBox 对象

TComboBox 对象(下拉式菜单)与 TListBox 对象非常类似，两者都可以将可能的数据项完全的列出来，其位于组件面板上的 Standard 页标签中。但事实上两者还是有些微不同的，例如 TComboBox 对象允许用户在运行时期输入数据，再者 TComboBox 对象只能选取一个指定的数据项等，因此我们可以将 TComboBox 视为 TListBox 与 TEdit 等对象的结合体。以下就 TComboBox 对象的特有属性加以说明。

- *Items* 属性

Items 属性是真正保存 TComboBox 对象数据项的属性，Items 属性是继承自 TString 对象，有关 TString 对象的详细操作方法请参考第 15.2 节。如果您想在设计时期决定 Items 属性值，只要在 Items 属性值字段上双击，通过“String List Editor”对话框进行编辑即可。

- *Text* 属性

与 TEdit 对象一样，Text 属性正是用户在 TComboBox 对象所选取的内容。

- *MaxLength* 属性

与 TEdit 对象一样，MaxLength 属性是允许用户输入数据的最大字符串长度。

- *DropDownCount* 属性

DropDownCount 属性是用来决定 TComboBox 对象下拉菜单的高度，也就是数据项的数量。

- *Style* 属性

Style 属性可以用来决定是否允许用户输入数据，如果某些数据项固定不变，最好不要让

用户输入，以确保数据的正确性。Style 属性包含有 csDropDown、csSimple、csDropDownList、csOwnerDrawFixed 及 csOwnerDrawVariable 等属性值。其中只有 csDropDown 与 csSimple 才允许用户输入，当 TComboBox 对象的 Style 属性值被设为 csSimple 时，此时与 TEdit 对象是完全一样的。



8-6

目的：本范例主要的目的是说明 TComboBox 对象常用的属性。

STEP 1　创建新的工程。

STEP 2　在 TForm 对象上加入三个 TLabel 对象、两个 TComboBox 对象及一个 TListBox 对象，其相关属性设置如表 8.4 所示。

表 8.4

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	中文数据
TLabel	Name	Label2
	Caption	英文数据
TLabel	Name	Label1
	Caption	您所选择的数据如下：
TComboBox	Name	ComboBox1
	Style	csDropDownList
	Items	星期一、星期二、星期三、星期四、星期五、星期六、星期日
TComboBox	Name	ComboBox2
	Style	csDropDownList
	Items	Monday、Tuesday、Wednesday、Thursday、Friday、Starurday、Sunday
TListBox	Name	Listbox1

STEP 3　在 ComboBox1 对象的 onChange 事件处理过程中加入以下代码：

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  //当用户选取中文数据，则出现对应的英文数据
  comboBox2.itemindex:=comboBox1.itemindex;
  ListBox1.Items.add(ComboBox1.Text);
end;
```

STEP 4　在 ComboBox2 对象的 onChange 事件处理过程中加入以下代码：

```
procedure TForm1.ComboBox2Change(Sender: TObject);
begin
```

```
    //当用户选取英文数据，则出现对应的中文数据
    comboBox1.itemindex:=comboBox2.itemindex;
    Listbox1.Items.add(ComboBox2.Text);
end;
```

5 保存工程。

6 按【F9】功能键运行工程，当工程运行后，只要您选择中文数据，系统便自动显示对应的英文数据，并记录所选取的数据项。反之，选择英文数据亦同，如图 8.19 所示。



图 8.19 TComboBox 对象应用范例

- *AutoComplete* 属性

当 AutoComplete 属性为 True 时，用户在 TComboBox 对象上输入某段字符串(英文字符串或中文字符串都适用)时，系统会自动跳至最符合条件的项目上。通常会配合 Sorted 属性为 True 一起使用。

- *AutoDropDown* 属性

AutoDropDown 属性可以决定用户在 TComboBox 对象上输入时是否显示下拉菜单，当 AutoDropDown 属性为 True 表示自动显示；反之，则不显示，如图 8.20 所示。

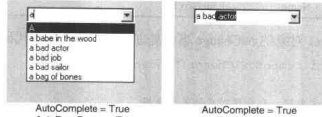


图 8.20 TComboBox 对象 AutoDropDown 属性

8.5 TComboBoxEx 对象

TComboBoxEx 对象可以说是 TComboBox 对象的豪华版，其具有与 TComboBox 相同的

功能，而且可以制作出更漂亮的下拉菜单，其位于组件面板上的 Win32 页标签中。以下仅介绍 TComboBoxEx 对象特有属性，至于其他与 TComboBox 对象相同的属性，请参考上一章节的详细说明。

8.5.1　TComboBoxEx 对象常用属性

TComboBoxEx 对象有很多属性与 TComboBox 对象相同，故以下仅对 TComboBoxEx 特有的属性加以说明。

Images 属性

TComboBoxEx 对象与其他对象相同，同样可以通过 TImagesList 对象来为每一个选项加入图标。

ItemsEx 属性

TComboBox 对象是通过 Items 属性来保存每一个数据项，因此 Items 属性对应于一个 TString 对象。与 TComboBox 对象不同的在于 TComboBoxEx 对象是通过 ItemsEx 属性来保存每一个数据项，而 ItemsEx 属性是对应于一个 TComboExItems 对象。我们将于下一个小节中详细说明有关 TComboExItems 对象的属性与操作方法。

StyleEx 属性

StyleEx 属性主要是用来决定每一个数据项的行为，包括：csExCaseSensitive、csExNoEditImage、csExNoEditImageIndent、csExNoSizeLimit 及 csExPathWordBreak 等。CsExCaseSensitive 代表搜索数据时是否将大小写视为不同；csExNoEditImage 代表数据项被显示时并不会显示其所对应的图片，也就是说此 TComboBoxEx 对象会忽略其 ImageIndex 属性值；csExNoEditImageIndent 代表数据项被显示时并不会作数据项的缩排显示，也就是说此 TComboBoxEx 对象会忽略其 Indent 属性值；csExNoSizeLimit 代表是否对用户改变画面大小作限制；csExPathWordBreak 则代表是否将“\”、“/”及“.”等符号视为键盘引导(keyboard navigation)字符。

8.5.2　TComboExItems 对象

在 TComboBoxEx 对象中的 ItemsEx 属性对应于一个 TComboExItems 对象，在 TComboExItems 对象中的 ComboItems 属性才是真正的包含每一个数据项，而每一个数据项都是一个 TComboExItem 对象。笔者示范下列三种方法以协助您获取 TComboBoxEx 对象中的每一个数据项。

```
//方法一
ComboBoxEx1.ItemsEx.ComboItems[i].Caption
//方法二
ComboBoxEx1.ItemsEx[i].Caption
//方法三
ComboBoxEx1.Items[i]
```

第一种方法是通过 ItemsEx 属性先获取 TComboExItems 对象，此对象包含所有的 TComboExItem 对象。您可以再通过顺序索引值从 TComboExItem 对象中获取每一个数据项的标题，也就是 Caption 属性。第二种方法其实是第一种方法的简略用法，您可以直接在 ItemsEx 属性根据顺序索引值获取每一个数据项的 Caption 属性值。第三种方法主要是通过 Items 属性以获取每一个数据项的内容值，其用法与 TComboBox 对象的 Items 属性完全相同。

您现在已经知道如何获取 TComboBoxEx 对象中的每一个数据项，接着，笔者将为您介绍，如何动态产生或删除数据项。与添加或删除数据项有关的操作方法包括 AddItem、Add、Insert、DeleteSelected 与 Delete 等，分述如下：

AddItem 操作方法

AddItem 操作方法可以为 TComboBoxEx 对象添加一个数据项，该函数有 Caption、ImageIndex、SelectedImageIndex、OverlayImageIndex、Indent 及 Data 等参数。

```
function AddItem(const Caption: String; const ImageIndex,
        SelectedImageIndex, OverlayImageIndex,
        Indent: Integer; Data: Pointer): TComboExItem;
```

Add 操作方法

当您调用 Add 操作方法时，系统会自动为 TComboBoxEx 对象附加一个空的数据项，此操作方法会返回一个 TComboExItem 对象。

```
function Add: TComboExItem;
```

Insert 操作方法

当您调用 Insert 操作方法时，系统会在指定的位置为 TComboBoxEx 对象插入一个空的数据项，此操作方法会返回一个 TComboExItem 对象。如果您指定的位置已存在，那么系统自动覆盖原来的数据项，其中的 Index 参数代表指定的数据项顺序位置。

```
function Insert(Index: Integer): TComboExItem;
```

Delete 操作方法

Delete 操作方法可以删除指定的数据项，其中的 Index 参数代表指定的数据项顺序位置、

```
procedure Delete(Index: Integer);
```

- *DeleteSelected* 操作方法

DeleteSelected 操作方法可以删除目前的数据项。

```
procedure DeleteSelected;
```

- *Clear* 操作方法

Clear 操作方法可以删除所有的数据项。

```
procedure Clear;
```

TComboExItem 是真正保存项目数据的对象，其常用的属性包括 Caption、ImageIndex、Indent、OverlayImageIndex、SelectedImageIndex 及 Data 等。

- *Caption* 属性

Caption 属性代表此数据项所要显示的内容。

- *ImageIndex* 属性

ImageIndex 属性代表此数据项所要显示的图片索引顺序，通常此属性必须配合 TImageList 对象一起使用。

- *Indent* 属性

Indent 属性代表此数据项缩排的次数。

- *OverlayImageIndex* 属性

OverlayImageIndex 属性代表此数据项所要显示的透明图片索引顺序，通常其对应的图形文件为黑白图片。一般来说，此属性必须配合 TImageList 对象一起使用。

- *SelectedImageIndex* 属性

ImageIndex 属性代表此数据项所要显示的图形文件，而 SelectedImageIndex 属性则是代表此数据项被点选时的显示图形文件。一般来说，此属性必须配合 TImageList 对象一起使用。

- *Data* 属性

Data 是一个非常特别的属性，其可对应于一个指针类型的变量。在实务上，每一个数据项可能还有其他需要表达或记录的数据，无奈 TComboExItem 对象所提供的属性有限，这个时候您就可以通过 Data 属性来保存其他所要记录的数据。Data 属性的详细使用方法请参考接下来的范例说明。

接着，笔者将为您示范如何在设计时期为 TComboBoxEx 对象创建数据项。至于如何在运行时期创建方式，请参考接下来的范例说明。

1. 在 TForm 对象上加入一个 TImageList 对象并为其加入几个图形文件，请参考范例 6-4 的详细示范。
2. 在TForm对象上加入一个TComboBoxEx对象并将其Images属性值设为ImageList1。
3. 通过对象检查器点选 TComboBoxEx 对象的 ItemEx 属性，以打开选项编辑器。
4. 您可以在选项编辑器中添加、修改或删除选项，如图 8.21 所示。

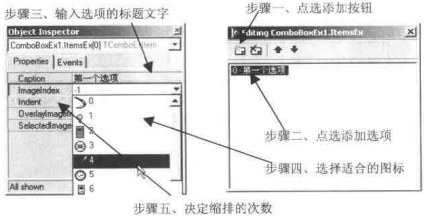


图 8.21 创建 TComboBoxEx 对象的数据项

范例 8-7

目的：本范例主要是说明 TComboBoxEx 对象相关属性与操作方法。

1. 创建新的工程。
2. 在 TForm 对象上加入一个 TLabel 对象、一个 TEdit 对象、一个 TMemo 对象、一个 TImageList 对象、一个 TComboBoxEx 对象及八个 TButton 对象，其相关属性设置如表 8.5 所示。

表 8.5

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	目前数据项个数：
TEdit	Name	Edit1
TMemo	Name	Memo1
TImageList	Name	ImageList1
TComboBoxEx	Name	ComboBoxEx1
	Images	ImageList1
	Style	csExDropDownList

续上表

对象	属性名称	属性设置值
TButton	Name	ButtonItemsAdd
	Caption	通过 Items 属性创建数据项
TButton	Name	ButtonInsert
	Caption	Insert
TButton	Name	ButtonAdd
	Caption	Add
TButton	Name	ButtonDisaplyAll
	Caption	Display All
TButton	Name	ButtonAddItem
	Caption	AddItem
TButton	Name	ButtonDelete1
	Caption	Delete1
TButton	Name	ButtonDelete2
	Caption	Delete3
TButton	Name	ButtonClear
	Caption	Clear

STEP 3 自定义一个指针(Pointer)数据类型，用于存储数据项的创建时间，其代码如下：

```
type
  //自定义一个结构的数据类型
  PMyRecord = ^TMyRecord;
  TMyRecord = record
    CreateTime : String; //创建的时间
  end;
```

STEP 4 在 ComboBoxEx1 对象的 onChange 事件处理过程中加入以下的代码：

```
procedure TForm1.ComboBoxEx1Change(Sender: TObject);
begin
  //显示目前数据项的 Data 属性值
  if ComboBoxEx1.ItemIndex <> -1 then
    Memo1.Lines.Append(PMyRecord(ComboBoxEx1.ItemsEx
                [ComboBoxEx1.ItemIndex].Data)^.CreateTime);
end;
```

STEP 5 在 ButtonItemsAdd 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonItemsAddClick(Sender: TObject);
var
  //声明指针变量
  MyRecordPointer: PMyRecord;
```

```
    i : Integer;
begin
    //创建指针变量
    New(MyRecordPointer);
    //赋值变量值(以目前时间为例)
    MyRecordPointer.CreateTime := DateTimeToStr(Now);
    //加入一个数据项
    i := ComboBoxEx1.Items.Add('Items.Add Item');
    ComboBoxEx1.ItemsEx.ComboItems[i].ImageIndex :=i mod 10;
    ComboBoxEx1.ItemsEx.ComboItems[i].Indent := i;
    ComboBoxEx1.ItemsEx.ComboItems[i].Data := MyRecordPointer;
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
end;
```

6 在 ButtonInsert 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonInsertClick(Sender: TObject);
var
    //声明指针变量
    MyRecordPointer: PMyRecord;
    tmp : TComboExItem;
begin
    //创建指针变量
    New(MyRecordPointer);
    //赋值变量值(以目前时间为例)
    MyRecordPointer.CreateTime := DateTimeToStr(Now);
    //在最后一个数据项后插入一个新的数据项
    tmp := ComboBoxEx1.ItemsEx.Insert(ComboBoxEx1.ItemsEx.Count);
    tmp.Caption := 'Inserted Item';
    tmp.Indent  := ComboBoxEx1.ItemsEx.Count mod 10;
    tmp.ImageIndex := tmp.Indent mod 10;
    tmp.SelectedImageIndex := -1;
    tmp.OverlayImageIndex  := -1;
    tmp.Data :=  MyRecordPointer;
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
end;
```

7 在 ButtonAdd 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonAddClick(Sender: TObject);
var
    //声明指针变量
    MyRecordPointer: PMyRecord;
```

```
    tmp : TComboExItem;
  begin
    //创建指针变量
    New(MyRecordPointer);
    //赋值变量值(以目前时间为例)
    MyRecordPointer.CreateTime := DateTimeToStr(Now);
    //加入一个新的数据项
    tmp := ComboBoxEx1.ItemsEx.Add;
    tmp.Caption := 'Added Item';
    tmp.Indent := ComboBoxEx1.ItemsEx.Count mod 10;
    tmp.ImageIndex := tmp.Indent mod 10;
    tmp.SelectedImageIndex := -1;
    tmp.OverlayImageIndex := -1;
    tmp.Data := MyRecordPointer;
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
  end;
```

8 在 ButtonDisaplyAll 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonDisaplyAllClick(Sender: TObject);
var
  I : Integer;
begin
  //清除 Memo1 对象内容
  Memo1.Clear;
  //获取所有的选项
  for i:=0 to ComboBoxEx1.ItemsEx.Count -1 do
    //显示内容
    Memo1.Lines.Append(ComboBoxEx1.ItemsEx.ComboItems[i].Caption);
end;
```

9 在 ButtonAddItem 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonAddItemClick(Sender: TObject);
var
  //声明指针变量
  MyRecordPointer: PMyRecord;
  Caption : String;
  ImageIndex, SelectedImageIndex, OverlayImageIndex , Indent : Integer;
begin
  //创建指针变量
  New(MyRecordPointer);
  //赋值变量值(以目前时间为例)
  MyRecordPointer.CreateTime := DateTimeToStr(Now);
```

```
    //加入新的数据项
    Caption := 'AddItem Item';
    Indent := ComboBoxEx1.ItemsEx.Count mod 10;
    ImageIndex := Indent mod 10;
    SelectedImageIndex := -1;
    OverlayImageIndex := -1;
    ComboBoxEx1.ItemsEx.AddItem(Caption,ImageIndex,
                        SelectedImageIndex,OverlayImageIndex,Indent,
                        MyRecordPointer);
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
end;
```

STEP 10 在 ButtonDelete1 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonDelete1Click(Sender: TObject);
begin
    //删除目前的数据
    ComboBoxEx1.DeleteSelected;
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
end;
```

STEP 11 在 ButtonDelete2 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonDelete2Click(Sender: TObject);
begin
    //删除第一个数据项
    ComboBoxEx1.ItemsEx.Delete(0);
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
end;
```

STEP 12 在 ButtonClear 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonClearClick(Sender: TObject);
begin
    //清除所有数据项
    ComboBoxEx1.Clear;
    //显示目前的数据项个数
    Edit1.Text := IntToStr(ComboBoxEx1.ItemsEx.Count);
end;
```

STEP 13 保存工程并按【F9】键运行工程，工程运行后，您可以使用“通过 Items 属性创建数据项”、“Insert”、“Add”及“AddItem”等按钮为 ComboBoxEx1 对象添加数据项。也可以利用“Delete 1”或“Delete 2”按钮分别删除目前的数据项或是第一个数据项。也可以使用“Clear”按钮清除 ComboBoxEx1 对象所有的数据项。当然，

您也可以使用“Display All”按钮来显示 ComboBoxEx1 对象的所有数据项。除此之外，当您选择 ComboBoxEx1 对象的某个数据项时，系统会将其 Data 属性值显示在 Memo1 对象上，如图 8.22 所示。



图 8.22　TComboBoxEx 对象范例

8.6　TRadioButton 对象

TRadioButton 对象通常会配合分类对象(如 TForm、TPanel、TgroupBox 等)。在某一个分类对象中的 TRadioButton 对象我们将其视为同一个组，在同一组、同一时间内所有的 TRadioButton 对象是具有互斥性的。换言之，也就是同一组的 TRadioButtons 对象同时只能有一个被选取，就像单选题一样。我们通常由 Checked 属性来判断某个 TRaionButton 对象是否被用户选取。

范例 8-8

目的：本范例主要的目的是说明 TRadioButton 对象常用的属性。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TLabel 对象、一个 TButton 对象及两个 TRadioButton 对象，其属性值设置如表 8.6 所示。

表 8.6

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	请问您的性别是：
TButton	Name	Button1
	Caption	确定
TRadioButton	Name	RadioButton1
	Caption	男生
	Checked	True
TRadioButton	Name	RadioButton2
	Caption	女生

步骤 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str:String;
begin
  if RadioButton1.Checked then
    Str:=RadioButton1.Caption
  else
    Str:=RadioButton2.Caption;
  ShowMessage('您的性别是:'+Str);
end;
```

步骤 4 保存工程。

步骤 5 按下【F9】功能键运行工程，当工程运行后，您可以按下任意的 TRadioButton 对象，再单击“确定”按钮，接着由程序判断您所选取的是哪一个选项，如图 8.23 所示。

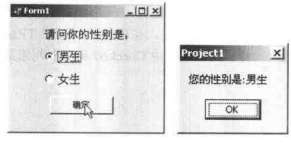


图 8.23 TRadioButton 对象应用范例

8.7 TCheckBox 对象

如果说 TRadioButton 是单选题的话，那么 TCheckBox 对象就是复选题了。TCheckBox 对象在应用上不一定要配合分类对象使用，但是若为了要有清晰的用户界面，您可以尽量使用分类对象。TCheckBox 对象除了有 Checked 属性可以判断用户是否选取外，更有 State 属性允许用户有更多的选择。State 属性通常配合着 AllowGrayed 属性一起使用。

范例 8-9

目的：本范例主要的目的是说明 TCheckBox 对象常用的属性。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TLabel 对象及六个 TCheckBox 对象，其相关属性设置如表 8.7 所示。

表 8.7

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	您的兴趣有哪些
TCheckBox	Name	CheckBoxSwimming
	Caption	游泳
TCheckBox	Name	CheckBoxSing
	Caption	唱歌
TCheckBox	Name	CheckBoxBall
	Caption	球类运动
	AllowGrayed	True
TCheckBox	Name	CheckBoxBall1
	Caption	篮球
TCheckBox	Name	CheckBoxBall2
	Caption	排球
TCheckBox	Name	CheckBoxBall3
	Caption	台球

步骤 3 在 CheckBoxBall 对象(球类运动)的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.CheckBoxBallClick(Sender: TObject);
begin
  //当 CheckBoxBall 被完全选取
  if CheckBoxBall.Checked then
    begin
      CheckBoxBall1.Checked:=True;
      CheckBoxBall2.Checked:=True;
      CheckBoxBall3.Checked:=True;
    end
   //当 CheckBoxBall 完全被取消选取
   else
    begin
      CheckBoxBall1.Checked:=False;
      CheckBoxBall2.Checked:=False;
      CheckBoxBall3.Checked:=False;
    end;
  //当 CheckBoxBall 的 State 属性为 cbGrayed,
 //则设置任一个选项为被选取
  if CheckBoxBall.State = cbGrayed then
    begin
      CheckBoxBall1.Checked:=True;
```

```
        CheckBoxBall2.Checked:=False;
        CheckBoxBall3.Checked:=False;
      end;
end;
```

步骤 4 在 CheckBoxBall1 对象(篮球)的 onClick 事件处理过程中加入以下的代码，并与 CheckBoxBall2(排球)及 CheckBoxBall3(台球)的事件处理过程共享。

```
procedure TForm1.CheckBoxBall1Click(Sender: TObject);
begin
    //假如每一种球类运动都被选取
    if (CheckBoxBall1.Checked) and (CheckBoxBall2.Checked)
       and (CheckBoxBall3.Checked) then
            CheckBoxBall.Checked:=True
    //仅选取某一种球类运动
    else if (CheckBoxBall1.Checked) OR
            (CheckBoxBall2.Checked) OR
            (CheckBoxBall3.Checked) then
            begin
                 //为了避免重复触发 CheckBoxBall 的 onClick
                 //事件，故先将其设为 nil
                CheckBoxBall.onClick:=nil;
               CheckBoxBall.State:=cbGrayed;
               CheckBoxBall.onClick:=CheckBoxBallClick;
            end
    //没有选取任何一种球类运动
    else  CheckBoxBall.Checked:=False;
end;
```

步骤 5 保存工程。

步骤 6 按下【F9】键运行工程，工程运行后，假如您只选取部分的球类运动，则 CheckBoxBall(球类运动)会出现灰色的被选取状态，如图 8.24 所示。



图 8.24 TCheckBox 对象应用范例

8.8　TCheckListBox 对象

TCheckListBox 对象是 TListBox 对象与 TCheckBox 对象的结合体。TCheckListBox 对象位于组件面板上的 Additional 页标签中。它同时具有 TListBox 对象及 TCheckBox 对象的所有属性与操作方法，相关的说明请参考本章 8.3 及 8.7 两个小节，如图 8.25 所示。



图 8.25　TCheckListBox 对象

以就仅就 TCheckListBox 对象常用的属性加以说明，其包括 Items、AllowGrayed、Checked、State、Selected、Flat 及 ItemEnabled 等，分述如下：

Items 属性

Items 属性是真正保存 TCheckListBox 对象数据项的属性，Items 属性继承自 TString 对象，有关 TString 对象请参考第 15.2 节的详细说明。

AllowGrayed 属性

当 AllowGrayed 属性值为 True，表示 TCheckListBox 对象中的每一个数据项都允许有 Grayed 的状态。也就是说，每一个数据项在“Checked(选取)”和“非 Checked(非选取)”之间另外存在一个“Grayed”的状态。有关 AllowGrayed 属性应用实例请参考范例 8-9 的详细介绍。

Checked 属性

当 Checked 属性为 True，表示该选项已被用户选取(表示有☑符号)，此属性与 TCheckBox 对象的 Checked 属性用法相同。

State 属性

State 属性可记录目前该选项的状态，包括：cbUnchecked、cbChecked 及 cbGrayed 等。cbUnchecked 代表该选项并未被选取；cbChecked 代表该选项已被选取；cbGrayed 代表该选项的状态正处理 cbUnchecked 与 cbChecked 之间。当 Checked 属性值为 True 时，State 属性值必为 cbChecked；反之，当 Checked 属性值为 False 时，State 属性值可能为 cbUnchecked 或 cbGrayed。

Selected 属性

当 Selected 属性为 True，表示该项目已被用户选取(反白数据项)，此属性与 TListBox 对象的 Selected 属性用法相同。

Flat 属性

Flat 属性是用来决定 TCheckListBox 对象是否具有 Flat 的特性，如图 8.26 所示。

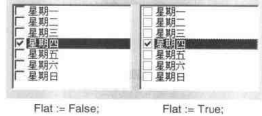


图 8.26 TCheckListBox 对象的 Flat 属性

ItemEnabled 属性

ItemEnabled 属性可以个别决定每一个数据项为 Enable 或 Disable。

范例 8-10

目的：本范例上要是示范如何使用 TCheckListBox 对象的常用属性。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TCheckListBox 对象、一个 TMemo 对象及一个 TButton 对象。

步骤 3 通过 TCheckListBox 对象的 Items 属性加入“星期一”到“星期日”等七个选项。

步骤 4 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码，这段代码主要的目的是将 CheckListBox1 对象设置为允许复选(MultiSelect)，并 Disable 其中的一个数据项。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //允许多条数据项同时选取
  CheckListBox1.MultiSelect := True;
  //Disable 某一个数据项
  CheckListBox1.ItemEnabled[3] := False;
end;
```

步骤 5 在 TButton 对象的 onClick 事件处理过程中加入以下的代码，这段代码主要目的是按照顺序查看每一个数据项的状态，这些状态包括 Seleced、Checked 及 State 等。

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    i : Integer;
    str : String;
begin
    //清除 Memo1 的内容
    Memo1.Clear;
    //按照顺序查看每一个选项的状态
    for i:= 0 to CheckListBox1.Items.Count -1 do begin
      str := '';
      if CheckListBox1.Selected[i] then str := str +
                                        Format('%15s',['Selected']);
      if CheckListBox1.Checked[i] then str := str +
                                        Format('%15s',['Checked']);
      if CheckListBox1.State[i] = cbUnchecked then str := str +
                                        Format('%15s',['cbUnchecked']);
      if CheckListBox1.State[i] = cbChecked then str := str +
                                        Format('%15s',['cbChecked' ]);
      if CheckListBox1.State[i] = cbGrayed then str := str +
                                        Format('%15s',['cbGrayed']);
      if CheckListBox1.ItemEnabled[i] then str := str +
                                        Format('%15s',['ItemEnabled']);
      Memo1.Lines.Append(CheckListBox1.Items[i] + Str);
    end;
end;
```

6 保存并运行工程，工程运行后，您可以任意点选 TCheckListBox 对象中的数据项。也可以按下 TButton 对象来查看每一个数据项的状态，如图 8.27 所示。

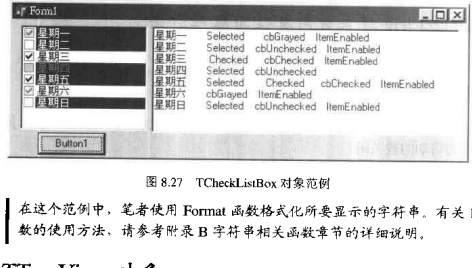


图 8.27　TCheckListBox 对象范例

注意　在这个范例中，笔者使用 Format 函数格式化所要显示的字符串。有关 Format 函数的使用方法，请参考附录 B 字符串相关函数章节的详细说明。

8.9　TTreeView 对象

TTreeView 对象又称为树状视图组件，它就像大家所熟知的“资源管理器”左半边的功

能，其位于组件面板上的 Win32 页标签中。在说明 TTreeView 对象之前您有几个名词是必须要了解的，例如 Root Node(根节点)、Parent Node(父节点)、Child Node(子节点)及 Leave Node(叶节点)等。这里所谓的“节点”都是继承至 TTreeNode 对象。

- *Root Node(根节点)*

根节点是每一个树状视图组件最顶层的节点。

- *Parent Node(父节点)、Child Node(子节点)及 Leave Node(叶节点)*

父节点与子节点是相对的，它们都是树状视图组件的一个节点。如果某个节点并不是最顶层的结点(根节点)，它必然会有一个父节点。如果某个节点没有子节点，那么它可以被称为 Leave Node(叶节点)，如图 8.28 所示。

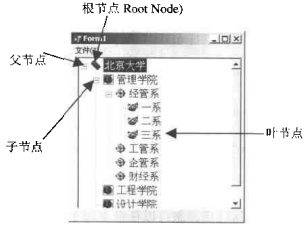


图 8.28 树状视图组件

8.9.1 TTreeView 对象常用属性与操作方法

以下就 TTreeView 对象(树状视图组件)各种常用的属性与操作方法加以说明：

- *AutoExpand 属性*

当 AutoExpand 属性被设为 True 时，只有被选取的节点才会显示出子节点，其他未被选取的节点都会自动的被关闭。

- *ChangeDelay 属性*

ChangeDelay 属性用来设置某节点被选取与发生 onChange 事件之间的时间。

- *Images 属性*

Images 属性用来设置 TTreeView 对象在各层次节点所要显示的图片列表，一般会配合 TImageList 对象使用。

Items 属性

Items 属性是用来设置 TTreeView 对象的内容值，它继承自 TTreeNode 对象，您可以在运行时期改变它，当然您也可以在设计时期利用“TreeView Items Editor”加以编辑，如图 8.29 所示。

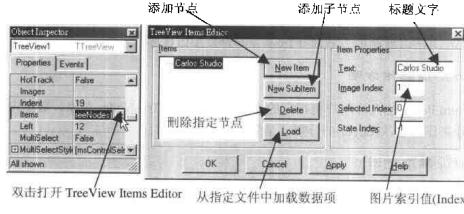


图 8.29　树状视图组件 Items 属性

SortType 属性

SortType 属性用来设置 AlphaShort 操作方法的排序依据。SortType 属性共有四种属性值 stNone(不排序)、stData(按数据类型排序)、stText(按照 Caption 属性排序)及 stBoth(按数据类型及 Caption 属性加以排序)。

ShowButtons 属性

ShowButtons 属性用来设置当某个节点包含有子节点是否出现“+”符号，或当子节点被延展(开启)是否出现“-”符号。ShowButtons 属性值为 True，表示会出现；反之，则不会出现。

ShowLines 属性

ShowLines 属性用来决定是否以“线段”将父节点与子节点加以连接。ShowLines 属性值为 True，表示会出现；反之，则不会出现。

RowSelect 属性

RowSelect 属性用来设置当某个节点被选取时，要以整行反白或部分反白显示。RowSelect 属性值为 True，表示反白整行；反之，反白部分。

Selected 属性

Selected 属性本身继承自 TTreeNode 对象，用来说明哪一个节点为被选取中的对象。

HideSelection 属性

HideSelection 属性用来设置当此 TTreeView 对象不具有焦点时(Focus)，是否维持以反白方式来显示目前所被选取的节点。当 HideSelection 为 False，表示维持以反白方式来显示；反之，则取消反白。

AlphaSort 操作方法

AlphaSort 方法以默认的方式对 TTreeView 对象加以排序。

LoadFromFile 操作方法

LoadFromFile 方法允许用户将指定的文件内容加载 TTreeNode 对象。数据内容必须以“Tab”符号来作为层次(Level)的分隔，请参考图 8.29 所示。

SaveToFile 操作方法

SaveToFile 方法允许用户将指定的 TTreeNode 对象内容保存到文件中。在保存过程中数据内容会自动以“Tab”符号来作为层次(Level)的分隔，如图 8.30 所示。

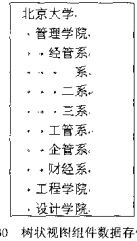


图 8.30 树状视图组件数据存储格式

8.9.2 *TTreeNode* 对象

如上所述，在 TTreeView 对象中每一个节点都是一个 TTreeNode 对象。因此如果您要添加一个节点、删除指定节点、对某个节点重新命名等都必须通过 TTreeNode 对象。以下就 TTreeNode 对象常用的属性及操作方法加以说明。

Count 属性

Count 属性主要是用来记录指定节点直接包含的子节点数。

Expanded 属性

Expanded 属性主要是用来记录指定节点是否被开启(Expaneded)。

您也可以使用“Display All”按钮来显示 ComboBoxEx1 对象的所有数据项。除此之外，当您选择 ComboBoxEx1 对象的某个数据项时，系统会将其 Data 属性值显示在 Memo1 对象上，如图 8.22 所示。



图 8.22　TComboBoxEx 对象范例

8.6　TRadioButton 对象

TRadioButton 对象通常会配合分类对象(如 TForm、TPanel、TgroupBox 等)。在某一个分类对象中的 TRadioButton 对象我们将其视为同一个组，在同一组、同一时间内所有的 TRadioButton 对象是具有互斥性的。换言之，也就是同一组的 TRadioButtons 对象同时只能有一个被选取，就像单选题一样。我们通常由 Checked 属性来判断某个 TRaionButton 对象是否被用户选取。

范例 8-8

目的：本范例主要的目的是说明 TRadioButton 对象常用的属性。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TLabel 对象、一个 TButton 对象及两个 TRadioButton 对象，其属性值设置如表 8.6 所示。

表 8.6

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	请问您的性别是：
TButton	Name	Button1
	Caption	确定
TRadioButton	Name	RadioButton1
	Caption	男生
	Checked	True
TRadioButton	Name	RadioButton2
	Caption	女生

- *Delete* 操作方法

Delete 方法允许用户删除指定的 TTreeNode 对象。

- *AddChild* 操作方法

AddChild 方法允许用户在指定的 TTreeNode 对象下添加一个子节点。

范例 8-11

目的：本范例主要是说明 TTreeView 对象常用的属性与操作方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TOpenDialog 对象、一个 TsaveDialog 对象、一个 TMainMenu 对象、一个 TpopupMenu 对象、一个 TImageList 对象及一个 TTreeView 对象，其相关属性设置如表 8.8 所示。

表 8.8

对象	属性名称	属性设置值
TOpenDialog	Name	OpenDialog1
TSaveDialog	Name	SaveDialog1
TMainMenu	Name	MainMenu1
TPopupMenu	Name	PopupMenu1
TImageList	Name	ImageList1
(加入至少为 TTreeView 对象可能层次的图片数量)		
TTreeView	Name	TreeView1
	PopupMenu	PopubMenu1
	Images	ImageList1
	Font.Size	12
	HideSelection	False
	Items	创建一个名为“Carlos Studio”的根节点，并指定 Image Index。

STEP 3 在 TpopupMenu 对象中加入四个选择项目，其设置如表 8.9 所示。

表 8.9

对象	属性名称	属性设置值
TMenuItem	Name	MenuADD
	Caption	添加子节点
TMenuItem	Name	MenuDel
	Caption	删除子节点

续上表

对象	属性名称	属性设置值
TMenuItem	Name	MenuRename
	Caption	重命名
TMenuItem	Name	MenuSort
	Caption	重新排序节点

STEP 4 在 TpopupMenu 对象“添加子节点”选项的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.MenuADDClick(Sender: TObject);
begin
//加入新的节点
with TreeView1.Items.AddChild(TreeView1.Selected,'新节点') do
      //指定图片
      ImageIndex:= TreeView1.Selected.ImageIndex-1;
end;
```

STEP 5 在 TpopupMenu 对象“删除了节点”项目的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.MenuDelClick(Sender: TObject);
begin
  //删除指定节点，先判断节点不是根节点
  if TreeView1.Selected.Level <> 0 then
     TreeView1.Items.Delete(TreeView1.Selected);
end;
```

STEP 6 在 TpopupMenu 对象“重命名”项目的 onClick 事件处理过程以下的代码：

```
procedure TForm1.MenuReNameClick(Sender: TObject);
begin
    //重命名被选取的节点
    TreeView1.Selected.EditText;
end;
```

STEP 7 在 TpopupMenu 对象“重新排序”项目的 onClick 事件处理过程以下的代码：

```
procedure TForm1.MenuSortClick(Sender: TObject);
begin
    //重新排序
    TreeView1.AlphaSort;
end;
```

STEP 8 在 TMainMenu 对象中加入四个选择项目，其设置如表 8.10 所示。

表 8.10

对象	属性名称	属性设置值
TMenuItem	Name	MenuFile
	Caption	文件

续上表

对象	属性名称	属性设置值
TMenuItem	Name	MenuOpen
	Caption	打开文件
TMenuItem	Name	MenuSave
	Caption	保存文件
TMenuItem	Name	MenuChild
	Caption	显示子节点数
TMenuItem	Name	MenuLevel
	Caption	显示目前层次(Level)

STEP 9 在 TMainMenu 对象“打开文件”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuOpenClick(Sender: TObject);
begin
    //打开文件
   if OpenDialog1.execute then
     Treeview1.LoadFromFile(OpenDialog1.FileName);
end;
```

STEP 10 在 TMainMenu 对象“保存文件”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuSaveClick(Sender: TObject);
begin
    //保存
    if SaveDialog1.Execute then
      Treeview1.SavetoFile(SaveDialog1.Filename);
end;
```

STEP 11 在 TMainMenu 对象“显示子节点数”项目的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuChildClick(Sender: TObject);
begin
    //显示直接子节点数目
    Showmessage('您目前所在节点的有' +
                        inttostr(TreeView1.Selected.count)+'个子节点');
end;
```

STEP 12 在 TMainMenu 对象“显示目前层次(Level)”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuLevelClick(Sender: TObject);
begin
    //显示直接子节点数目
```

```
    Showmessage('您目前所在节点是第' +
                                inttostr(TreeView1.Selected.Level)+'层');
end;
```

(13) 保存工程。

(14) 按【F9】功能键运行工程，工程运行后您可以添加、删除或修改节点，甚至利用“打开文件”或“保存文件”来存取您的数据，如图 8.31 所示。



图 8.31　树状视图组件应用范例

8.9.3　TTreeView 对象与数据库

如果您是 Delphi 的初学者，笔者建议您先略过本小节，等您对数据库有相当程度的了解之后，再回来练习本小节所介绍的范例。有关 Delphi 6 与数据库的连接请参考本书第 17 章的详细说明。上一个小节中，笔者示范如何将数据写入文件与如何将文件的内容转为 TTreeView 对象的节点数据。事实上，这些都是 TTreeView 对象本身所提供的操作方法，应该不会有太高的困难度。但是，当您的数据格式很复杂，必须动用 TTreeView 对象的 Data 属性来记录数据时，上一小节所示范的方法是无法满足此需求的，面对复杂的数据结构时，我想数据库会是一个不错的保存方式。本小节将以实例的方式来说明如何将 TTreeView 对象的节点数据保存到数据库，或是将数据库的内容转为 TTreeView 对象的节点数据。

以下笔者将示范如何结合 Access 2000 数据库制作一个树状的网址管理程序，当然，这个程序的结构非常简单，您可以自行扩充功能以符合需求。在制作这个程序之前，我们必须先决定数据库的表结构，这个表的名称为 myTTreeView，并包含 ParentNodeID、NodeID、Caption 及 URL 等字段。ParentNodeID 字段代表父节点代码；NodeID 字段代表该节点的代码；Caption 字段代表节点标题；URL 字段代表该节点所对应的网址。其中 NodeID 字段为主键，也就是说每一个节点代码是惟一的，不会重复，如图 8.32 所示。

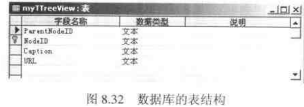


图 8.32　数据库的表结构

数据库创建后，请按照下列步骤即可完成一个简易的树状网址管理程序。

范例 8-12

目的：本范例主要是说明如何使用 TTreeView 对象创建一个简易的树状网址管理程序。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TImageList 对象并仿照范例 8-11 加入图形文件。

步骤 3 在 TForm 对象中加入一个 TTreeView 对象并仿照范例 8-11 为其创建一个名为 Carlos Studio 的根节点。

步骤 4 在 TForm 对象中加入一个 TWebBrowser 对象，此对象将作为显示网址的浏览器。

步骤 5 另外在 TForm 对象上加入一个 TPopupMenu 对象、一个 TLabel 对象及一个 TEdit 对象。除此之外，笔者为了画面的设计而加入几个 TPanel 对象，因为这些 TPanel 对象并不会影响程序功能，因此您可自行决定是否套用。

步骤 6 在 TPopupMenu 对象中加入“添加子节点”的选项，相关属性设置如表 8.11 所示。

表 8.11

对象	属性名称	属性设置值
TMenuItem	Name	MenuADD
	Caption	添加子节点

步骤 7 将 TTreeView 对象的 PopupMenu 属性设为 PopupMenu1、Images 属性设为 ImageList1。

步骤 8 声明一个自定义的指针数据类型，其包括父节点、节点代码、标题文字及网址等字段。

```
type
  //自定义一个结构的数据类型
  PMyRecord = ^TMyRecord;
  TMyRecord = record
    ParentNodeID : String;   //父节点
    NodeID : String;         //节点代码
    Caption : String;        //标题文字
    URL : String;            //网址
  end;
```

步骤 9 除了系统默认的 TForm 对象外，另外从菜单栏中的“File/ New/ Form”加入另一个 TForm 对象，其相关属性设置如表 8.12 所示。

表 8.12

对象	属性名称	属性设置值
TForm	Name	Form2
	Caption	添加节点
	Position	poScreenCenter

Step 10 在 Form2 对象上加入两个 TLabel、两个 TEdit 及两个 TBitBtn 对象，其相关属性设置如表 8.13 所示。

表 8.13

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	标题文字
TLabel	Name	Label2
	Caption	网址
TEdit	Name	EditCaption
	Text	(空白)
TEdit	Name	EditURL
	Text	(空白)
TBitBtn	Name	BitBtn1
	Kind	bkOK
	Enabled	False
TBitBtn	Name	BitBtn2
	Kind	bkCancel

Step 11 在 EditCaption 对象的 onChange 事件处理过程中加入以下代码，并将此事件处理过程与 EditURL 对象共享。这段代码主要的目的是通过 BitBtn1 的 Enabled 属性来控制获取数据的正确性，也就是说“标题文字”与“网址”缺一不可。

```
procedure TForm2.EditCaptionChange(Sender: TObject);
begin
   BitBtn1.Enabled :=  (EditCaption.Text <> '')
                                        and (EditURL.Text <> '');
end;
```

Step 12 在 Form1 对象上加入一个 TADOConnection 数据库连接对象。您可以点选 ConectionString 属性旁的按钮，通过“连接信息编辑器”设置相关的连接信息，相关设置请参考第 18 章的详细说明。除了 ConectionString 属性外，其余的属性设置如表 8.14 所示。

表 8.14

对象	属性名称	属性设置值
TADOConnection	Name	ADOConnection1
	ConectionString	(略)
	LoginPrompt	False
	Connected	True

Step 13 在 Form1 对象上加入一个 TADOTable 对象，其相关属性设置如表 8.15 所示。

表 8.15

对象	属性名称	属性设置值
TADOTable	Name	ADOTable1
	Connection	ADOConnection1
	TableName	myTTreeView
	Connected	True

STEP 14 为 Form1 的 MenuADD 对象的 onClick 事件处理过程加入“显示输入画面”、“将数据保存到 TTreeView 对象”及“保存到数据库”等功能的代码，详细的代码如下所示：

```
procedure TForm1.MenuADDClick(Sender: TObject);
var
  //声明指针变量
  MyRecordPointer: PMyRecord;
begin
  with TForm2.Create(Application) do begin
    try
      ShowModal;
    finally
      //假如用户单击 OK 按钮，则将数据加以保存
      if ModalResult = mrOK then begin
        //创建指针变量
        New(MyRecordPointer);
        //加入新的节点
        with TreeView1.Items.AddChild(TreeView1.Selected,
            EditCaption.Text) do begin
            //指定图片
            ImageIndex:= Level + 1;
            Selectedindex := Level + 1;
            //指定指针变量的内容
            MyRecordPointer.ParentNodeID :=
                  PMyRecord(TreeView1.Selected.Data)^.NodeID;
            MyRecordPointer.NodeID :=
                  PMyRecord(TreeView1.Selected.Data)^.NodeID
                   + '-' + IntToStr(TreeView1.Selected.Count);
            MyRecordPointer.Caption := EditCaption.Text;;
            MyRecordPointer.URL := EditURL.Text;
            //设置节点的 Data 属性值
            Data := MyRecordPointer;
        end;
        //将数据保存到数据库
        ADOTable1.Append;
```

```
            ADOTable1.FieldByName('ParentNodeID').AsString :=
                  PMyRecord(TreeView1.Selected.Data)^.NodeID;
            ADOTable1.FieldByName('NodeID').AsString :=
                  PMyRecord(TreeView1.Selected.Data)^.NodeID +
                  '-' + IntToStr(TreeView1.Selected.Count);
            ADOTable1.FieldByName('Caption').AsString :=
                  EditCaption.Text;
            ADOTable1.FieldByName('URL').AsString :=
                  EditURL.Text;
            ADOTable1.Post;
          end;
          Free;
        end;
    end;
end;
```

15 在 TreeView1 对象的 onClick 事件处理过程中加入以下的代码，这段代码将通过 WebBrowser1 对象显示并打开指定的网址。

```
procedure TForm1.TreeView1Click(Sender: TObject);
begin
  if TreeView1.Selected.Data <> nil then begin
    //显示所要连接的网址
    Edit1.Text := PMyRecord(TreeView1.Selected.Data)^.URL;
    //进行连接
    WebBrowser1.Navigate(PMyRecord(TreeView1.
Selected.Data)^.URL);
  end;
end;
```

16 经过漫长的说明步骤，这个程序即将完成。但接下来的工作才是本范例所要介绍的重点，没错，就是要将数据库的内容转为 TTreeView 对象节点数据。为此，我们特别编写一个子程序来完成此项工作，其内容如下所示：

```
procedure TForm1.CreateSubNodes(myNode :TTreeNode);
var
  //声明指针变量
  MyRecordPointer: PMyRecord;
  tmpDataSet : TADODataSet;
  tmpNode : TTreeNode;
  SQLStr : string;
begin
  //创建 DataSet
  tmpDataSet := TADODataSet.Create(nil);
```

```
    //组合 SQL 指令，找出指定节点的所有的子节点
    SQLStr := ' ParentNodeID = ''' +
                              PMyRecord(myNode.Data)^.NodeID + '''';
    //连接数据库
    tmpDataSet.Connection := ADOConnection1;
    tmpDataSet.CursorType := ctStatic;
    tmpDataSet.CursorLocation := clUseClient;
    tmpDataSet.CommandText := 'select * from  myTTreeView
                                 where ' + SQLStr + ' Order by NodeID ';
    tmpDataSet.Active := True;
    tmpDataSet.First;
    While Not tmpDataSet.Eof do begin
       //创建子节点
       tmpNode := TreeView1.Items.AddChild(myNode,
                             tmpDataSet.FieldValues['Caption']);
       //设置图片属性
       tmpNode.ImageIndex := tmpNode.Level + 1;
       tmpNode.SelectedIndex := tmpNode.Level + 1;
       //创建指针变量
       New(MyRecordPointer);
       //将相关数据保存到 tmpNode
       MyRecordPointer.ParentNodeID :=
                              tmpDataSet.FieldValues['ParentNodeID'];
       MyRecordPointer.NodeID :=
                              tmpDataSet.FieldValues['NodeID'];
       MyRecordPointer.Caption :=
                              tmpDataSet.FieldValues['Caption'];
       MyRecordPointer.URL := tmpDataSet.FieldValues['URL'];
       tmpNode.Data :=  MyRecordPointer;
       //创建子节点(以递归的方式调用)
       CreateSubNodes(tmpNode);
       tmpDataSet.Next;
    end;
    //关闭与释放 DataSet
    tmpDataSet.Active := False;
    tmpDataSet.Free;
end;
```

注意

在这个子程序中，主要可分为“从数据库中获取指定父节点的所有子节点”、“将子节点数据加到 TTreeView 对象，并将其余的字段数据写入指针变量中”、“以递归的方式再调用子程序本身，其参数为新创建的子节点”及“关闭与释放 TADODataSet 对象”等子功能。通过这种递归的调用方式才能轻易地将数据库的内容转为 TTreeView 对象的节点数据。

步骤 17 任何用户自定义函数都必须声明，因此请为上一步骤所创建的子程序进行声明。

```
public
  { Public declarations }
  //自定义函数
  procedure CreateSubNodes(myNode :TTreeNode);
end;
```

步骤 18 在 Form1 对象的 onCreate 事件处理过程调用 CreateSubNodes 子程序将数据库的内容转为 TTreeView 对象的节点数据。除此之外，根节点是在设计时期通过“TreeView Items Editor”编辑的，因此并没有记录其 Data 属性的内容值，所以我们必须额外以程序的方式加以修改。其代码如下所示：

```
procedure TForm1.FormCreate(Sender: TObject);
var
  //声明指针变量
  MyRecordPointer: PMyRecord;
begin
  //创建指针变量
  New(MyRecordPointer);
  //设置根节点的指针变量内容
  MyRecordPointer.ParentNodeID := '';
  MyRecordPointer.NodeID := 'Root';
  MyRecordPointer.Caption := TreeView1.Items[0].Text;
  MyRecordPointer.URL := '';
  TreeView1.Items[0].Data := MyRecordPointer;
  //将数据库的数据写到 TTreeView 对象中
  CreateSubNodes(TreeView1.Items[0]);
end;
```

步骤 19 保存并按【F9】键运行工程。工程运行后，系统会自动加载数据库中现有的数据，并且允许您添加自己的节点。运行结果如图 8.33 所示。

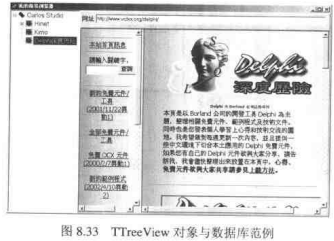


图 8.33　TTreeView 对象与数据库范例

8.10 TListView 对象

如果说 TTreeView 对象是文件总管的左半边，那么 TListView 对象就是文件总管的右半边。TListView 与资源管理器一样，提供了四种不同的显示方式，如大图标(vsIcon)、小图标(vsSmallIcon)、列表(vsList)及详细资料(vsReport)等。以下就 TListView 对象常用的属性与操作方法加以说明。

Items 属性

Items 属性是真正保存 TListView 对象数据项的属性，Items 属性是继承自 TString 对象，有关 TString 对象的详细操作方法请参考第 15.2 节。至于运行时期改变 Items 属性值常用的操作方法请参考第 9.1.2 节。如果您想在设计时期决定 Items 属性值，只要在 Items 属性值字段上双击，通过“String List Editor”对话框进行编辑即可。

Checkboxes 属性

CheckBoxes 属性用来决定在 TListView 对象中的每一个数据项前面是否出现一个 Checkbox 对象，以供用户选取。当 CheckBoxes 属性值为 True 时，此 TListView 对象就像 TCheckListBox 对象一样具有 Checkbox 功能。

ViewStyle 属性

ViewStyle 属性用来决定 TListView 对象显示的方式，共有四种不同的显示方式，如大图标(vsIcon)、小图标(vsSmallIcon)、列表(vsList)及详细资料(vsReport)等。

Columns 属性

您可以利用点选 Columns 属性，开启 TListView 对象的字段编辑器，在设计时期决定 TListView 对象所要出现的字段名称、宽度及字段个数等，如图 8.34 所示。

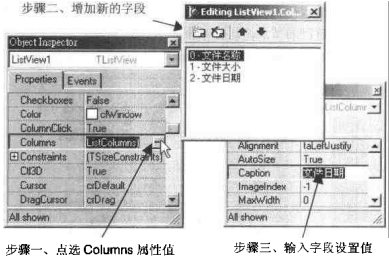


图 8.34 TListView 对象的字段编辑器

ColumnClick 属性

如果您设置 TListView 对象的 ViewStyle 属性的详细资料(vsReport)时，此时 TListView 对象的最上面会出现 Column(字段)，例如资源管理器中的“名称”、“大小”、“类型”及“修改的日期”等按钮。如果您又将 ColumnClick 属性设为 True 时，这些按钮就允许用户按下，当用户按下这些按钮时，会触发 ColumnClick 的事件。我们通常利用这个事件来进行不同的排序工作，其事件处理过程声明如下：

```
Procedure TForm1.ListView1ColumnClick(Sender: TObject;Column: TListColumn);
```

FlatScrollBars 属性

FlatScrollBars 属性决定 TListView 对象是否要以平滑式的 ScrollBar 出现。当 FlagScrollBars 属性值为 True，表示 TListView 对象的 ScrollBar 是平滑式；反之，则为一般的 ScrollBar。

GridLines 属性

GridLines 属性决定 TListView 对象是否显示网格线。当 GridLines 属性值为 True，表示 TListView 对象要显示网格线；反之，则隐藏网格线。

IconOptions 属性

IconOptions 是一个复合的属性值，其包括 Arrangement、AutoArrange 及 WrapText 等。Arrangement 属性是代表排列的方式，有向上对齐(iaTop)及向左对齐(iaLeft)。AutoArrange 属性代表是否自动排列，WrapText 属性决定是否将数据换行显示。

范例 8-13

目的：本范例主要配合使用文件处理相关对象(TFileListBox、TFilterComboBox、TDirectoryListBox 及 TDriveComboBox 等)用以说明 TListView 对象常用的属性与操作方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TDriveComboBox、一个 TDirectoryListbox 对象、一个 TFilterComboBox 对象、一个 TFileListBox 对象、一个 TListView 对象、一个 TPopupMenu 对象、两个 TImageList 对象及两个 TLabel 对象。其相关属性说明如表 8.16 所示。

表 8.16

对象	属性名称	属性设置值
TImageList	Name	ImageList1
	(加入 BigA.bmp 及 BigB.bmp)	

续上表

对象	属性名称	属性设置值
TImageList	Name	ImageList2
	(加入 SmallA.bmp 及 SmallB.bmp)	
TLabel	Name	Label1
	Caption	路径:
TLabel	Name	LabelPathname
	Caption	空白
TDriveComboBox	Name	DriveComboBox1
	DirList	DirectoryListBox1
TDirectoryListbox	Name	DirectoryListBox1
	DirLabel	LabelPathName
	FileList	FileListBox1
TFilterComboBox	Name	FilterComboBox1
	FileList	FileListBox1
	Filter	All files (*.*)\|*.*\|
		Text Files(*.txt)\|*.txt
TFileListBox	Name	FileListBox1
TListView	Name	ListView1
	ColumnClick	False
	Columns	文件名称、文件大小、文件日期
	LargeImages	ImageList1
	ShowColumnHeaders	True
	SmallImages	ImageList2
	SortType	stBoth
	ViewStyle	vsReport
	PopupMenu	PopupMenu1
TPopupMenu	Name	PopupMenu
TMenuItem	Name	MenuIcon
	Caption	大图标
TMenuItem	Name	MenuSmallIcon
	Caption	小图标
TMenuItem	Name	MenuList
	Caption	清单
TMenuItem	Name	MenuReport
	Caption	详细资料

步骤 3 在 TDirectoryListBox 对象的 onChange 事件处理过程加入以下的代码:

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
var
  NewItem:TListItem;  //新的数据项
  str,StrPath:String;
  SRec:TSearchRec;
  Flag:Integer;
begin
   //清除原有的数据项
   ListView1.Items.Clear;

   //获取正确的搜索条件
   Str:=FilterComboBox1.Text;
   Str:=(copy(str,pos('(',str)+1,pos(')',str)-pos('(',str)-1));

   //获取正确的目录(再加上搜索条件)
   if Length(DirectoryListBox1.Directory)<>3 then
      StrPath:=DirectoryListBox1.Directory+Str
   else
      StrPath:=DirectoryListBox1.Directory+'\'+Str;

   //找到第一条数据
   Flag:=FindFirst(StrPath,faAnyFile,SRec);

   //先判断是否找到了第一条数据
   if Flag=0 then
   begin
     //将找到的数据项加入TListView
     NewItem:=ListView1.Items.Add;
     NewItem.Caption:=SRec.Name;  //文件名
     NewItem.SubItems.Add(IntToStr(SRec.Size)); //文件大小
     NewItem.SubItems.Add(DateTimeToStr(
                    FileDateToDateTime(SRec.Time)));//时间
     if SRec.Attr = 16 then NewItem.ImageIndex:=1; //是目录

     //找下一条数据
     while (FindNext(SRec) = 0) do
     begin
        NewItem:=ListView1.Items.Add;
        NewItem.Caption:=SRec.Name;
        NewItem.SubItems.Add(IntToStr(SRec.Size));
        NewItem.SubItems.Add(DateTimeToStr(
                                FileDateToDateTime(SRec.Time)));
        if SRec.Attr = 16 then NewItem.ImageIndex:=1;
```

```
    end;
  end;

  //重新排序
  ListView1.AlphaSort;
end;
```

Step 4 在 TForm 对象的 onActive 事件处理过程中加入以下的代码：

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  //程序一启动就将数据显现在 TListView 对象上
  DirectoryListBox1Change(nil);
end;
```

Step 5 在 TPopupMenu 对象“大图标”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuIconClick(Sender: TObject);
begin
    //大图标
    ListView1.ViewStyle:=vsIcon;
end;
```

Step 6 在 TPopupMenu 对象“小图标”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuSmallIconClick(Sender: TObject);
begin
    //小图标
    ListView1.ViewStyle:=vsSmallIcon;
end;
```

Step 7 在 TPopupMenu 对象“列表”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuListClick(Sender: TObject);
begin
    //列表
    ListView1.ViewStyle:=vsList;
end;
```

Step 8 在 TPopupMenu 对象“详细资料”选项的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.MenuReportClick(Sender: TObject);
begin
    //详细资料
    ListView1.ViewStyle:=vsReport;
end;
```

Step 9 保存工程。

Step 10 按【F9】键运行工程。工程运行后，您可以打开任意的目录，此时系统会将该目录

的数据显示在 TListView 对象中，如图 8.35 所示。



图 8.35　TListView 对象综合范例

第9章

Delphi 6 分类对象

本章目标

本章将为各位介绍 Delphi 6 的分类对象，当我们在设计应用程序时，分类对象可以将画面按功能及种类加以分隔，让画面看起来更干净更协调等。这些分类对象包括 TGroupBox、TRadioGroup、TPanel、TScrollBox、TTabControl、TPageControl 及 THeaderControl 等。

本章主题

9.1 TgroupBox 对象

9.2 TRadioGroup 对象

9.3 TPancl 对象

9.4 TScrollBox 对象

9.5 TPageControl 对象

9.6 THeaderControl 对象

9.1 TGroupBox 对象

TGroupBox 对象，位于组件面板上的 Stanard 页标签中，常被我们用来分隔功能相似或同一主题的选项。它最常被使用的属性是 Caption。在图 9.1 中可以看出在适度的使用 TGroupBox 对象后，可以达到意想不到的效果。除了视觉效果外，TGroupBox 对象也可以将 TRadioButton 对象加以分类，例如图 9.1 中的“个人基本资料”与“个人爱好”等。

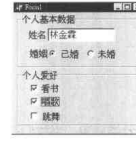


图 9.1 TGroupBox 的应用范例

9.2 TRadioGroup 对象

记得 TRadioButton 对象吗?如果有些忘记的话，那么您可以看看图 9.1 中的“已婚”及“未婚”两个选项，它就是 TRadionButton 对象。在这之前我们都是先创建一个分类对象(如 TForm 对象)，再加上 TRadioButton 对象便它们成为一个群组。接着笔者将为大家介绍如何在 TRadioGroup 对象(位于组件面板上的 Stanard 页标签)中创建一个或多个 TRadioButton 对象。TRadioGroup 对象与我们以前所使用的方法最大的不同点在于您不用一个一个的加入 TRadionButton 对象，而是设定相关属性即可自动产生 TRadionButton 对象。以下就将 TRadioGroup 对象常用的属性加以说明。

Caption 属性

Caption 属性与 TGroupBox 对象的 Capton 属性是完全相同的，它代表此分类对象的标题。

Items 属性

Items 属性是用来决定在 TGroupBox 对象所出现的选项标题名称。您可以点选 Items 字段以打开“String List Editor”对话框，通过“String List Editor”对话框来编辑所有的项目名称。由于 Items 属性是继承自 TString 对象，因此您可以使用 TString 对象的各项操作方法，如添加(Add)、删除(Delete)、插入(Insert)、交换(Exchange)等，详细说明请参考第 15.2 节。

ItemIndex 属性

ItemIndex 属性是用来决定或记录哪一个选项是被选取，其值由 0 算起。如果您不想任何一个选项被选取，只要将 ItemIndex 属性设为－1 即可。

Columns 属性

Columns 属性是用来决定 TRadioGroup 对象显示的字段数。在图 9.2 中就是 Columns 属性值为 3 的范例。



图 9.2　TGroupBox 的应用范例

范例 9-1

目的：本范例主要是说明 TRadioBox 对象的使用方法。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TRadioGroup 对象及五个 TButton 对象，其相关属性设置如表 9.1 所示。

表 9.1

对象	属性名称	属性设置值
TRadioGroup	Name	RadioGroup1
	ItemIndex	2
	Items	选项一、选项二、选项三
TButton	Name	ButtonAppend
	Caption	新增
TButton	Name	ButtonDelete
	Caption	删除
TButton	Name	ButtonInsert
	Caption	插入

续上表

对象	属性名称	属性设置值
TButton	Name	ButtonExchange
	Caption	交换
TButton	Name	ButtonOK
	Caption	确定

步骤 3 在 ButtonAppend 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonAppendClick(Sender: TObject);
var
  DefStr:String;
begin
  //添加一个选项(最后一个选项)
  DefStr:='未命名';
  if InputQuery('请输入选项名称','选项名称',DefStr) then
    RadioGroup1.Items.Add(DefStr);
end;
```

步骤 4 在 ButtonDelete 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
begin
  //删除一个选项
  if RadioGroup1.ItemIndex > -1 then
    RadioGroup1.Items.Delete(RadioGroup1.ItemIndex);
end;
```

步骤 5 在 ButtonInsert 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonInsertClick(Sender: TObject);
var
   DefStr:String;
   I:Integer;
begin
  //当没有任何选项被选取时 ItemIndex 为-1，因此必须转为 0
  if RadioGroup1.ItemIndex = -1 then
    I:=0
  else
    I:=RadioGroup1.ItemIndex;

  //插入一个选项
```

```
    DefStr:='未命名';
    if InputQuery('请输入选项名称','选项名称',DefStr) then
      RadioGroup1.Items.Insert(I,DefStr);
  end;
```

步骤6 在ButtonExchange对象的onClick事件处理过程中加入以下的代码：

```
  procedure TForm1.ButtonExchangeClick(Sender: TObject);
  var
     DefStr:String;
     I:Integer;
  begin
    //当没有任何选项被选取时ItemIndex为-1，因此必须转为0
    if RadioGroup1.ItemIndex = -1 then
      I:=0
    else
      I:=RadioGroup1.ItemIndex;

    //指定交换的选项
    DefStr:='0';
    if InputQuery('指定交换的选项','交换选项的代号',DefStr) then
      RadioGroup1.Items.Exchange(I,StrToInt(DefStr));
  end;
```

步骤7 在ButtonOK对象的onClick事件处理过程中加入以下的代码：

```
  procedure TForm1.ButtonOkClick(Sender: TObject);
  var
    Str:String;
  begin
    //获取用户所选取的选项
    Str:='目前有'+IntToStr(RadioGroup1.Items.Count)+
         '选项'+#10#13;
    if RadioGroup1.ItemIndex = -1 then
      Str:=Str+'您并未选择任何一个选项'
    else
      begin
        Str:=Str+'您所选择的选项代号:'+IntToStr(RadioGroup1.
                  ItemIndex) +#10#13;
        Str:=Str+'您所选择的选项名称:'+RadioGroup1.Items
                  [RadioGroup1.ItemIndex];
      end;
    ShowMessage(Str);
  end;
```

步骤 8 保存工程。

步骤 9 按【F9】功能键运行工程。工程运行后您可以动态的加入、删除及交换数据项，也可以取得目前您所选取的选项名称，如图 9.3 所示。

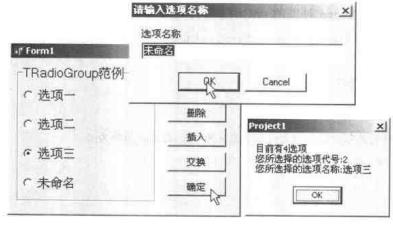


图 9.3 TRadioGroup 的应用范例

9.3 TPanel 对象

TPanel 对象，位于组件面板上的 Stanard 页标签中，也是常用的分类对象，通常是用来组合其他对象，以利画面设计。以下就 TPanel 对象各个常用的属性加以说明。

- *Caption 属性*

Caption 属性是代表 TPanel 对象的标题文字。

- *Align 属性*

Align 属性是用来决定 TPanel 对象在 TForm 对象上的位置。假如 Align 属性值为非 alTop 时，TPanel 对象置于 TForm 对象的最上端。并且会随着 TForm 对象的改变大小而改变 Tpanel 对象。Align 属性共有六种可能的值（见图 9.4），说明如表 9.2 所示。

表 9.2

Align 属性值	说明
AlNone	代表 TPanel 对象的位置由设计时期决定，而非固定位置
alTop	代表 TPanel 对象会固定在 TForm 对象的上方
atBottom	代表 TPanel 对象会固定在 TForm 对象的下方
alLeft	代表 TPanel 对象会固定在 TForm 对象的左方
alRight	代表 TPanel 对象会固定在 TForm 对象的右方
alClient	代表 TPanel 对象会覆盖 TForm 对象的全部空间

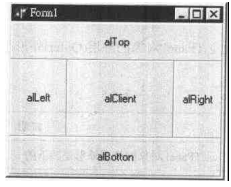


图 9.4　TPanel 对象的 Align 属性

Alignment 属性

Align 属性是代表 TPanel 对象的位置，而 Alignment 属性是用来决定标题文字(Caption 属性)的位置（见图 9.5），如表 9.3 所示。

表 9.3

Alignment 属性值	说明
taLeftJustify	标题文字向左对齐
taRightJustify	标题文字向右对齐
taCenter	标题文字向中对齐



图 9.5　TPanel 对象的 Alignment 属性

BevelInner 属性

BevelInner 属性是用来决定 TPanel 对象的内层(Inner)外形类型，其可能的属性值如表 9.4 所示。

表 9.4

BevelInner 属性值	说明
bvNone	表示 TPanel 对象的内层外形是凹下的
bvLowered	表示 TPanel 对象的内层外形是凸出的
bvRaised	表示 TPanel 对象的内层外形是无效果的

BevelOuter 属性

BevelOuter 属性是用来决定 TPanel 对象的外层(Outer)外形类型，其可能的属性值如表 9.5 所示。

表 9.5

BevelOuter 属性值	说明
BvNone	表示 TPanel 对象的外层外形是凹下的
BvLowered	表示 TPanel 对象的外层外形是凸出的
BvRaised	表示 TPanel 对象的外层外形是无效果的

BorderWidth 属性

BorderWidth 属性是用来决定 TPanel 对象外层(Outer)的宽度，当数字越大表示越宽。

BevelWidth 属性

BevelWidth 属性是用来决定 TPanel 对象的内层(Inner)及外层(Outer)之间的间距，当数字越大表示间距越宽。

BorderStyle 属性

BorderStyle 属性是用来决定 TPanel 对象有没有外框，当 BorderStyle 属性值为 bsSingle 表示 TPanel 对象有外框；反之，当 BorderStyle 属性值为 bsNone 表示 TPanel 对象没有外框，如图 9.6 所示。

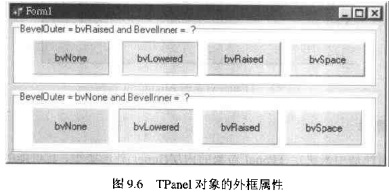


图 9.6 TPanel 对象的外框属性

Enabled 属性

当 TPanel 对象的 Enabled 属性被设置 False 时，表示在 TPanel 上的所有对象的 Enabled 属性也同时被设置 False。

9.4 TScrollBox 对象

TScrollBox 对象(位于组件面板上的 Additional 页标签中)在 TForm 对象上占了固定的范围，但是 TScrollBox 对象却可以加入比本身还要大的对象。当 TScrollBox 加入太多或太大的对象时，TScrollBox 对象会产生水平及垂直的滚动条，以便让使用者看到该对象的全貌。以下就 TScrollBox 对象各项常用的属性加以说明。

AutoScroll 属性

AutoScroll 属性是用来决定当 TScrollBox 加入太多或太大的对象时，TScrollBox 对象是否会自动产生对应的滚动条，以便让使用者看到该对象的全貌。当 AutoScroll 属性值为 True，表示 TScrollBox 对象会自动产生对应的滚动条。反之，则不会自动产生对应的滚动条。

AutoSize 属性

AutoSize 属性是用来决定当 TScrollBox 加入对象时，TScrollBox 对象是否会自动决定 TScrollBox 的大小。

VertScrollBar 与 HorzScrollBar 属性

前面曾经提过，当 TscrollBar 对象的 AutoScroll 属性值为 True 时，TscrollBar 对象会根据情况自动产生水平滚动条(VertScrollBar)与垂直滚动条(HorzScrollBar)。而这两个滚动条是继承至 TcontrolScrollBar 对象，因此表 9.6 是对对象常用的属性加以说明。

表 9.6

TcontrolScrollBar 对象属性值	说明
ButtonSize	代表滚动条两端按钮(◀、▶)的大小
Increment	代表滚动条两端按钮的移动量
Position	代表滚动条目前的位置
Size	代表滚动条的宽度
Smooth	决定是否将此滚动条设置平滑效果
Style	代表滚动条的样式，包括 ssRegular、ssFlat 及 ssHotTrack 等



9-2

目的：本范例主要是说明 TScrollBox 对象的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TScrollBox 对象、一个 TImage 对象、一个 TOpenPictureDialog 对象及一个 TButton 对象，其相关属性设置如表 9.7 所示。

表 9.7

对象	属性名称	属性设置值
TScrollBox	Name	ScrollBox1
	AutoScroll	True
TImage	Name	Image1
	AutoSize	True
TOpenPictureDialog	Name	OpenPictureDialog1
TButton	Name	ButtonOpen
	Caption	Open

步骤 3 在 TButton 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
begin
  //打开一个图片文件
  if OpenPictureDialog1.Execute then
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
```

步骤 4 保存工程。

步骤 5 按下【F9】功能键运行工程。工程运行后，当您加载比 TScrollBox 对象还要大的图片时，TScrollBox 会自动出现对应的滚动条，如图 9.7 所示：



图 9.7 TScrollBox 对象应用范例

9.5 TPageControl 对象

TPageControl 对象(位于组件面板上的 Win32 页标签中)是 Delphi 6 一个非常重要的分类对象。TPageControl 对象在应用程序的使用频率是非常高的，例如 Windows 95/98/NT/2000 等显示器调整对话框便是一个例子，如图 9.8 所示。

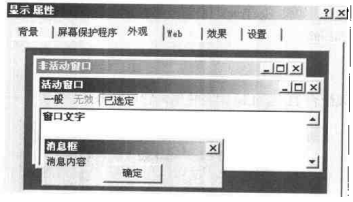


图 9.8　TPageControl 对象的应用

9.5.1　TPageControl 对象属性

TPageControl 对象可以利用有限的画面空间，表现出非常多的信息。再者，像一些安装向导也都可以利用 TPageControl 对象来完成。在说明各项常用的属性与事件之前，我们必须先来谈谈如何为 TPageControl 对象增加新的页？这里所谓的“页”，就是“TTabSheet”对象(详细说明请参考第 9.5.2 节)。要增加新的“页”其实很简单的，只要先在 TForm 对象上加入一个 TPageControl 对象（位于对象面板的 Win32 页标签中），并只在 TPageControl 对象上按右键，再选择“New Page”即可。如果您要切换到下一页，您可以选择“Next Page”，如果您想回到上一页，您可以选择“Previous Page”，如图 9.9 所示。

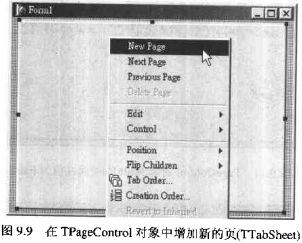


图 9.9　在 TPageControl 对象中增加新的页(TTabSheet)

以下就 TPageControl 对象常用的属性与事件加以说明：

- *ActivePage* 属性

ActivePage 属性用来决定目前活动中的页(TTabSheet)。

- *HotTrack* 属性

HotTrack 属性用来决定当鼠标指到“页标签”上会不会自动出现 Highlighted，也就是页

标签文字会自动变颜色。如果 HotTrack 属性值为 True，表示会自动出现 HighLight，反之，则代表不会出现 HighLight。

Images 属性

Images 属性必须配合 TImageList 对象使用，它可以为每一页提供一个图形化的标题。

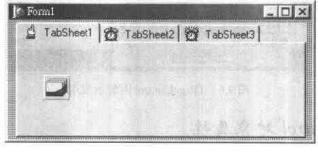


图 9.10　TPageControl 对象 Images 属性

MultiLine 属性

MultiLine 属性用来决定 TPageControl 对象标题是否要以多行(MultiLine)的形式出现。当 MultiLine 属性值为 True 时，表示 TPageControl 对象标题会以多行(MultiLine)的形式出现；反之，则以单行形式出现，如图 9.11 和图 9.12 所示。

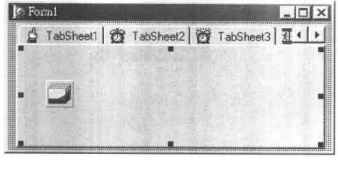


图 9.11　PageControl 对象 MultiLine 属性值为 False

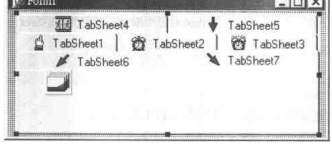


图 9.12　TPageControl 对象 MultiLine 属性值为 True

Style 属性

Style 属性用来决定 TPageControl 对象的页标签类型，Style 有三种可能的属性值：tsTabs、tsButtons 及 tsFlatButtons。分述如表 9.8 所示。

表 9.8

Style 可能的属性值	说明
tsTabs	代表一般的页标签形式，如图 9.12 所示
tsButtons	代表页标签以按钮的形式出现，如图 9.13 所示
tsFlatButtons	代表页标签以 FlatButton 的形式出现，如图 9.14 所示

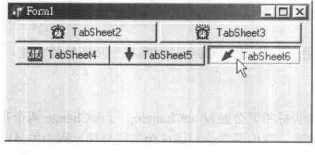


图 9.13　TPageControl 对象 Style 属性值为 tsButtons

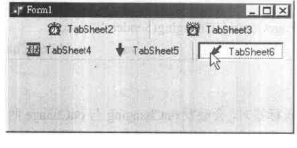


图 9.14　TPageControl 对象 Style 属性值为 tsFlatButtons

TabPosition 属性

TabPosition 属性是用来决定 TPageControl 对象页标签(Tab)出现的位置，TabPosition 属性有 **tpTop**、**tpBottom**、**tpLeft** 及 **tpRight** 等四种可能值（见图 9.15），分述如表 9.9 所示。

表 9.9

TabPosition 属性可能值	说明
tpTop	页标签出现在上方(默认值)
tpBottom	页标签出现在下方，如图 9.15 所示
tpLeft	页标标签现在左方，如图 9.15 所示
tpRight	页标签出现在右方，如图 9.15 所示

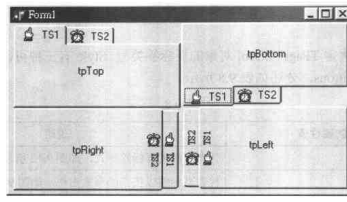


图 9.15 TPageControl 对象 TabPosition 各种属性值

- *TabWidth* 属性

TabWidth 属性是用来决定 TPageControl 对象每个页标签的宽度。

- *OnChanging* 事件

当使用者按下某个页标签时会触发 onChanging 与 onChange 两个事件。OnChanging 是发生在页标签发生改变之前，我们通常会利用这个事件处理过程中的 AllowChange 参数来决定是否可以离开目前的页标签，当 AllowChange 为 True，表示可以离开目前的页标签；反之，表示不可以离开目前的页标签。其函数声明原型如下：

```
procedure TForm1.PageControl1Changing(Sender: TObject;
                                      var      AllowChange: Boolean);
```

- *OnChange* 事件

当使用者按下某个页标签时，会触发 onChanging 与 onChange 两个事件。OnChange 是发生在页标发生改变之后，我们通常会利用这个事件处理过程来进行每一个页(TTabSheet)显示之前的设置。其函数声明原型如下：

```
procedure TForm1.PageControl1Change(Sender: TObject);
```

范例 9-3

目的：本范例主要是说明 TPageControl 对象的使用方法。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TPageControl 对象，并为 TPageConrol 对象添加三个页(TabSheet)，其相关设置如表 9.10 所示。

表 9.10

对象	属性名称	属性设置值
TPageControl	Name	PageControl1
TTabSheet	Name	TabSheet1
	Caption	第一页
TTabSheet	Name	TabSheet2
	Caption	第二页
TTabSheet	Name	TabSheet3
	Caption	第三页

STEP 3 在 TPageControl 对象的 onChanging 事件处理过程中加入以下的代码：

```
procedure TForm1.PageControl1Chang_ng(Sender: TObject;
  var AllowChange: Boolean);
begin
  //利用 onChanging 事件来判断是否离开目前的页(TabSheet)
  if MessageDlg('您是否真的要离开'+
            TPageControl(Sender).ActivePage.Caption,
            mtConfirmation,[mbYES,mbNO],0)=mrYes then
    AllowChange:=True
  else
    AllowChange:=False;
end;
```

STEP 4 在 TPageControl 对象的 onChange 事件处理过程中加入以下的代码：

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  //利用 onChange 事件处理过程来决定页标签改变后的相关设置
  ShowMessage('您现在进入'+
                    TPageControl(Sender).ActivePage.Caption);
end;
```

STEP 5 保存工程。

STEP 6 按【F9】键运行工程。工程运行后，只要您想进入某一个页，系统会先询问您是否离开目前的页。如果顺利进入某个页，系统也会提醒目前所进入的页，如图 9.16 所示。

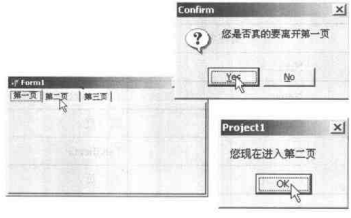


图 9.16 TPageControl 对象范例

9.5.2 TTabSheet 对象

TPageControl 对象中每一页就是一个 TTabSheet 对象，TTabSheet 对象也是真正放置其他对象的地方。以下就 TTabSheet 对象各项常用的属性加以说明：

- *Caption* 属性

Caption 属性用来设置每一页的标题文字。

- *ImageIndex* 属性

ImageIndex 属性用来设置每一页在标题中所出现的图片索引，此属性必须配合 TPageControl 对象的 Images 属性一起使用。

- *PageIndex* 属性

PageIndex 属性用来设置目前此 TTabSheet 对象在 TPageControl 对象出现的顺序。

- *TabVisible* 属性

TabVisible 属性用来决定目前此 TTabSheet 对象是否出现页标签，一般来说，如果您要制作向导(Wizard)，就必须将此 TabVisible 属性设为 False。



9-4

目的：本范例主要利用 TPageControl 对象来制作简单的向导(Wizard)。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TPanel 对象及一个 TPageControl 对象，并为 TPageControl 对象加入三个 TTabSheet 对象，其相关属性设置如表 9.11 所示。

表 9.11

对象	属性名称	属性设置值
TPanel	Name	Panel1
	Align	alBottom
	Caption	(空白)
TPageControl	Name	PageCntrl
	Align	alClient
TTabSheet	Name	TabSheet1
	TabVisible	False
	Caption	TabSheet1
TTabSheet	Name	TabSheet2
	TabVisible	False
	Caption	TabSheet2
TTabSheet	Name	TabSheet3
	TabVisible	False
	Caption	TabSheet3

STEP 3 在 TPanel 对象中加入三个 TButton 对象，其相关属性设置如表 9.12 所示。

表 9.12

对象	属性名称	属性设置值
TButton	Name	ButtonPrior
	Caption	上一页
TButton	Name	ButtonNext
	Caption	下一页
TButton	Name	ButtonCancel
	Caption	结束

STEP 4 在 TabSheet1 对象中加入一个 TImage 对象及两个 TLabel 对象，其相关属性设置如表 9.13 所示。

表 9.13

对象	属性名称	属性设置值
TImage	Name	Image1
	Picture	/Borland Shared/Images/Splash /256Color/Handshak.bmp
	Stretch	True
TLabel	Name	Label1
	Caption	欢迎光临
TLabel	Name	Label2
	Caption	客户基本数据建档向导

步骤 5 设计窗体如图 9.17 所示。



图 9.17 TabSheet1 的相关对象摆放位置

步骤 6 在 TabSheet2 对象中加入四个 TLabel 对象、一个 TMaskEdit 对象、一个 TEdit 对象、两个 TRadioButton 及一个 TDateTimePicker 对象，其相关属性设置如表 9.14 所示。

表 9.14

对象	属性名称	属性设置值
TLabel	Name	LabelNO
	Caption	客户编号
TLabel	Name	LabelName
	Caption	客户姓名
TLabel	Name	LabelSex
	Caption	客户性别:
TLabel	Name	LabelDate
	Caption	第一次交易时间

续上表

对象	属性名称	属性设置值
TMaskEdit	Name	MaskEditNo
	MaskEdit	999;0;_
TEdit	Name	EditName
	Text	(空白)
TRadioButton	Name	RadioButtonSex1
	Caption	男
	Checked	True
TRadioButton	Name	RadioButtonSex2
	Caption	女
TDateTimePicker	Name	DateTimePicker1

7 设计窗体如图 9.18 所示。

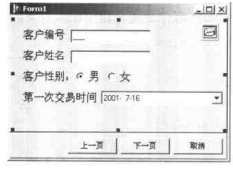


图 9.18　TabSheet2 的相关对象摆放位置

8 在 TabSheet3 对象中加入一个 TLabel 对象、一个 TScrollBox 对象、一个 TImage 对象、一个 TButton 对象及一个 TOpenPictureDialog 对象，其相关属性设置如表 9.15 所示。

表 9.15

对象	属性名称	属性设置值
TLabel	Name	LabelPicture
	Caption	客户相片
TScrollBox	Name	ScrollBox
TImage	Name	Image2
	AutoSize	True

续上表

对象	属性名称	属性设置值
TButton	Name	ButtonLoad
	Caption	加载图片...
TOpenPictureDialog	Name	OpenPictureDialog1

步骤 9 设计窗体如图 9.19 所示。

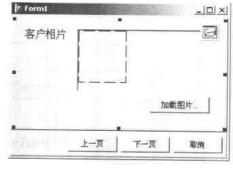


图 9.19 TabSheet3 的相关对象摆放位置

步骤 10 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    //设置开始的 TabSheet
    PageControl1.ActivePage:=TabSheet1;
    ButtonPrior.Enabled:=False;
end;
```

步骤 11 为 TabSheet3 对象中的 ButtonLoad(加载图片)对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonLoadClick(Sender: TObject);
begin
  //加载图片
  if OpenPictureDialog1.Execute then
     Image2.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
```

步骤 12 在 TPanel 对象的 ButtonPrior(上一页)对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPriorClick(Sender: TObject);
begin
   //回上一页
   if PageControl1.ActivePage=TabSheet3 then
```

```
      begin
        PageControl1.ActivePage:=TabSheet2;
        ButtonPrior.Enabled:=True;
      end
    else if PageControl1.ActivePage=TabSheet2 then
      begin
       PageControl1.ActivePage:=TabSheet1;
       //目前为第一页时必须将上一页 disable
       ButtonPrior.Enabled:=False;
      end;
  end;
```

(13) 在 TPanel 对象的 ButtonNex(下一页)对象的 onClick 事件处理过程中加入以下的代码:

```
procedure TForm1.ButtonNextClick(Sender: TObject);
var
  Flag:Boolean;
begin
  //到下一页
  if PageControl1.ActivePage=TabSheet1 then
    begin
      PageControl1.ActivePage:=TabSheet2;
      ButtonPrior.Enabled:=True;
    end
  else if PageControl1.ActivePage=TabSheet2 then
    begin
     //加入判断条件，例如某些字段一定要有值
     Flag:=(Length(MaskEditNO.Text)=3) and
            (Length(EditName.Text)>0);
     if Flag then
       begin
         PageControl1.ActivePage:=TabSheet3;
         //目前为最后一页时必须将下一页的标题改为"确定"
         ButtonNext.caption:='确定';
       end
     else
       ShowMessage('请输入正确的客户编号及客户姓名!!');
    end
  else
    begin
      //此时数据已收集完毕，你可以将其存到数据库中
      ShowMessage('客户基本数据已建档完毕');
```

```
        MaskEditNO.Text:='';
        EditName.Text:='';
        DateTimePicker1.DateTime:=now;
        PageControl1.ActivePage:=TabSheet1;
      end;
end;
```

步骤14 在 TPanel 对象的 ButtonCancel(结束)对象的 onClick 事件处理过程中加入以下的代码:

```
procedure TForm1.ButtonCancelClick(Sender: TObject);
begin
  //结束
  Close;
end;
```

步骤15 保存工程。

步骤16 按【F9】功能键运行工程。工程运行后，您可以测试这个向导。当您没有正确输入客户编号及姓名时，是无法跳到下一页继续其他工作，如此一来可以保持数据的有效性(如图 9.20 所示)。您更可以配合数据库的使用将数据或客户照片(如图9.21 所示)加以保存，或比较客户编号是否重复等检查。

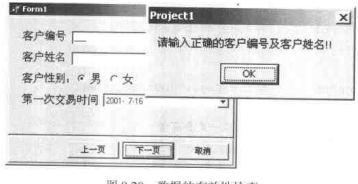


图 9.20 数据的有效性检查



图 9.21 加载图片

9.6　THeaderControl 对象

THeaderControl 对象(表头控件)是一个拥有不同区段(Section)的可视化控件，其位于组件面板上的 Win32 页标签中，在 THeaderControl 对象的每个区段可以显示文字或图片，并且允许用户在运行时期使用鼠标来调整每个区段的大小。以下就针对 THeaderControl 对象的各项常用属性与方法加以说明。

Section 属性

Section 要算是 TheaderControl 对象最重要的属性。一个 TheaderControl 对象可以拥有多个 Section。您可以点选 Section 属性字段打开“区段(Section)编辑器”来添加、修改或删除 Section。如图 9.22 所示。

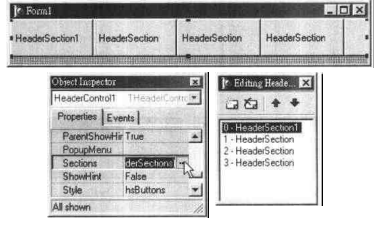


图 9.22　Section 编辑器

Section 属性继承自 TheaderSections 对象，表 9.16 将相关的属性加以说明。

表 9.16

TheaderSections 对象的属性	说明
Alignment	Alignment 属性用来决定 Section 对象文字的对齐方式，包括 taLeftJustify(左对齐)、taRightJustify(右对齐)、 taCenter(居中对齐)等
AllowClick	AllowClick 属性决定此 Section 对象是否会触发 onClick 事件。当 AllowClick 为 True，表示此 Section 会触发 onClick 事件
AutoSize	AutoSize 属性决定此 Section 对象是否会自动根据文字或图片来决定 Section 对象宽度
MinWidth	MinWidth 属性用来设置此 Section 对象的最小宽度
MaxWidth	MaxWidth 属性用来设置此 Section 对象的最大宽度
Width	Width 属性用来设置此 Section 对象目前的宽度

续上表

THeaderSections 对象的属性	说明
Style	Style 属性用来设置此 Section 对象的类型，包括 hsText 及 hsOwnerDraw 等。当 Style 属性被设为 hsText 时，Text 属性才有效。反之，Style 属性被设为 hsOwnerDraw 时，Text 属性是无效的。但却可以配合 Canvas 属性，为此 Seciton 加载图片(详细请参考范例 11-5)
Text	Text 属性是用来设置此 Section 对象的文字标题，但 Style 属性值必须为 hsText 时，才有效

- *Canvas 属性*

Canvas 是 THeaderControl 对象一个运行时期的属性，通过使用 Canvas 属性可以在运行时期加载图片或文字。但必须配合 Section 属性值为 hsOwnerDraw 时才有效。

- *Style 属性*

Style 属性是继承自 THeaderStyle 对象，Style 属性包括 hsButtons 及 hsFlat 两种属性值，如图 9.23 所示。



图 9.23 THeaderSection 对象的 Style 属性

- *OnDrawSection 事件*

当 THeaderSection 对象被创建(Create)时，会触发 onDrawSection 事件。OnDrawSection 事件只会针对 Style 属性为 hsOwnerDraw 的 Section 对象进行加载图片或其他的处理与设置。其函数声明原型如下：

```
procedure TForm1.HeaderControlDrawSection(HeaderControl:
        THeaderControl;   Section: THeaderSection; const Rect: TRect;
        Pressed: Boolean);
```

- *OnSectionClick 事件*

当 Section 对象被用户按下鼠标左键时，会触发 onSectionClick 事件，其函数声明原型如下：

```
procedure TForm1.HeaderControlSectionClick(HeaderControl:
                THeaderControl;   Section: THeaderSection);
```

范例 9-5

目的：本范例主要是说明 THeaderSection 对象的使用方法。

Step 1 创建新的工程。

Step 2 在 TForm 对象中加入一个 THeaderControl 对象及一个 TButton 对象，并为 THeaderControl 对象加入两个 THeaderSection 对象，其相关属性如表 9.17 所示。

表 9.17

对象	属性名称	属性设置值
THeaderControl	Name	HeaderControl1
	Height	57
	Align	alTop
THeaderControl.Section[0]	AutoSize	False
	Style	hsOwnerDraw
	Text	标题一
	Width	80
THeaderControl.Section[1]	AutoSize	False
	Style	hsText
	Text	标题二
	Width	80
TButton	Name	ButtonAdd
	Caption	Add Seciton...

Step 3 在 THeaderControl 对象的 onDrawSection 事件处理过程中加入以下的代码：

```
procedure  TForm1.HeaderControlDrawSection(HeaderControl:  THeaderControl;
Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
var
  MyPic:TPicture;
begin
  //只会针对 Style 属性为 hsOwnerDraw 的 Section 对象
  //进行加载图片或其他的处理与设置
  Try
    Mypic:=TPicture.Create;
    MyPic.Icon.LoadFromFile('earth.ico');
    HeaderControl.Canvas.Draw(Rect.Left,Rect.Top,MyPic.Icon);
```

```
    HeaderControl.Canvas.TextOut(Rect.Left,Rect.Top+
                                 MyPic.Icon.Height,'图片标题');
  Finally
    MyPic.Free;
  end;
  //设置文字颜色
  if Pressed then
    HeaderControl.Font.Color := clRed
  else
    HeaderControl.Font.Color := clBlue;
end;
```

步骤 4 在 THeaderControl 对象的 onSectionClick 事件处理过程中加入以下的代码：

```
procedure TForm1.HeaderControlSectionClick(HeaderControl: THeaderControl;
Section: THeaderSection);
begin
  //显示目前所按下的 section.Text
  ShowMessage('您现在所按下的是:',Section.Text);
end;
```

步骤 5 在 ButtonAdd 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonAddClick(Sender: TObject);
var
  HeaderSection: THeaderSection;
  Str:String;
  Flag:Boolean;
begin
  //加入一个新的 Section
  Str:='未命名';
  Flag:=InputQuery('请输入新的标题','标题',Str);
  if Flag then
    begin
      HeaderSection := HeaderControl.Sections.Add;
      with HeaderSection do
      begin
        HeaderSection.Text := Str;
        HeaderSection.MaxWidth:=120;
        HeaderSection.Width:=80;
        HeaderSection.MinWidth := length(Str) * Font.Size;
      end;
    end;
end;
```

6 保存工程。

7 按【F9】功能键运行工程。工程运行后，您会发现“表头一”自动的加入一个图片及蓝色的文字，当您按下鼠标左键时会显示您目前所按下的 Section 名称(Text 属性)，如图 9.24 所示。

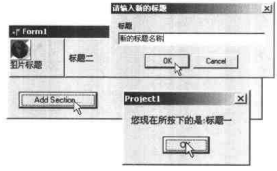


图 9.24　THeaderSection 对象范例

第10章

Delphi 6 常用对话框（一）

本章目标

本章将为各位介绍 Delphi 6 常用的对话框。Delphi 6 常用的对话框有一、二十个，本章所要介绍的对话框包括 ShowMessage、MessageDlg、MessageBox、InputBox、InputQuery、OpenDialog、SaveDialog、OpenPictureDialog、SavePictureDialog、FontDialog、ColorDialog、PrintDialog、PrintSetUpDialog、FindDialog 及 ReplaceDialog 等。

本章主题

10.1 ShowMessage 对话框

10.2 MessageDlg 对话框

10.3 MessageBox 对话框

10.4 InputBox 对话框

10.5 InputQuery 对话框

10.6 OpenDialog 与 SaveDialog 对话框

10.7 OpenPictureDialog 与 SavePictureDialog 对话框

10.8 FontDialog 与 ColorDialog 对话框

10.9 PrintDialog 与 PrintSetUpDialog 对话框

10.10 FindDialog 与 ReplaceDialog 对话框

10.1 ShowMessage 对话框

ShowMessage 对话框是 Delphi 6 中最简单的消息对话框，前面几章的范例大都以它作为显示消息的对话框。现在我们就来看看这个对话框的声明。

```
procedure ShowMessage(const Msg: string);
```

从这个对话框的声明函数中可以了解，您只要以 String 类型的数据为参数即可，如果您想在 ShowMessage 对话框中显示多笔文字，请使用“#10”作为换行符。“#10”代表 ASCII 10，也就是所谓的 Linefeed 字符。以下就以实例来说明 ShowMessage 对话框。

范例 10-1

目的：本范例主要是说明 ShowMessage 对话框的使用方法。

(1) 创建一个新的工程。

(2) 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ShowMessage('Delphi 常用的对话框' + #10 + 'ShowMessage');
end;
```

(3) 保存工程。

(4) 按【F9】功能键运行工程。工程运行后，对话框即可出现，如图 10.1 所示。



图 10.1 ShowMessage 对话框



如果您想在 ShowMessage 对话框中显示非字符串的内容，请先进行数据类型的转换即可，相关的转换函数请参考附录相关章节的详细介绍。例如您可以使用 DataTimeToStr 函数将日期时间类型转为字符串类型，或者使用 FloatToStr 函数将浮点数类型转为字符串类型。

10.2 MessageDlg 对话框

ShowMessage 对话框只能用于显示消息，至于用户是无法进行任何选择的，反正就是一定要单击“OK”按钮，程序才会继续运行下去。因此如果您希望程序能显示消息并且可以询

问用户的意见，我想 ShowMessage 对话框是无法办到的，这个时候您可以考虑使用 MessageDlg 对话框。其对话框的声明如下：

```
function MessageDlg(const Msg: string; DlgType: TMsgDlgType;
    Buttons: TMsgDlgButtons; HelpCtx: Longint): Word;
```

从这个声明您可以看出 MessageDlg 函数共有四个参数及一个返回值。参数一是用户所要显示的文字内容、参数二是对话框的类型、参数三是在对话框中所要出现的按钮、参数四是联机帮助说明。惟一的返回值是返回用户所按下的按钮。以下就对话框的类型及可以使用的按钮加以说明。

MessageDlg 对话框类型（见表 10.1）

表 10.1

对话框类型	图样	用途
mtWarning		警告信息
mtError		错误信息
mtInformation		显示信息
mtConfirmation		使用确认
mtCustom		用户自定义类型



当 MessageDlg 对话框类型被指定为 mtCustom 时，表示此对话框不仅没有图标，而且其对话框的标题文字也自动设置为工程的名称。

MessageDlg 对话框按钮（见表 10.2）

表 10.2

对话框类型	图样	返回值
mbYes	Yes	mrYes
mbNo	No	mrNo
mbOK	OK	mrOK
mbCancel	Cancel	mrCancel
mbAbort	Abort	mrAbort
mbRetry	Retry	mrRetry
mbIgnore	Ignore	mrIgnore
mbAll	All	mrAll
mbNoToAll	No to All	mrNoToAll
mbYesToAll	Yes to All	mrYesToAll
mbHelp	Help	mrHelp

范例 10-2

目的：本范例主要是说明 MessageDlg 对话框的使用方法。

步骤 1 创建一个新的工程。

步骤 2 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    if MessageDlg('您要继续运行吗',mtConfirmation,
                        [mbYES,mbNO],0)=mrNO then
     Halt;
end;
```

步骤 3 保存工程。

步骤 4 按【F9】功能键运行工程。工程运行后，会出现“您要继续运行吗”的对话框，假如您单击“Yes”按钮，则程序继续运行；反之，程序则立即结束，如图 10.2 所示。



图 10.2 MessageDlg 对话框

注意

在范例中我们使用 Halt 过程强制结束系统，Halt 过程被定义在 System 单元文件中，您不需要特别引用此单元文件。所谓强制结束系统，也就是以非正常的方式结束系统。在 Halt 过程中可以加入不同的参数值(exit code)，以利标识程序的结束点。如果 Halt 过程是以非正常的方式结束系统，那么正常的方式又是什么呢？在单一的 TForm 对象的系统中，您可以调用 TForm 对象的 Close 操作方法来结束它，也可以使用 TApplication 对象的 Terminate 操作方法。但是在多个 TForm 对象的系统中，而且结束点又不是在主要的 TForm 对象时，那么您必须使用 TApplication 对象的 Terminate 操作方法。

为了方便您使用 MessageDlg 对话框，Delphi 6 将几个常用的按钮以常数的方式加以定义。这些常数包括：mbYesNoCancel、mbYesAllNoAllCancel、mbOKCancel、mbAbortRetryIgnore 及 mbAbortIgnore 等。其中 mbYesNoCancel 代表 mbYes、mbNo 与 mbCancel；mbYesAllNoAllCancel 代表 mbYes、mbYesToAll、mbNo、mbNoToAll 与 mbCancel；mbOKCancel 代表 mbOK 与 mbCancel；mbAbortRetryIgnore 代表 mbAbort、mbRetry 与 mbIgnore；mbAbortIgnore 代表 mbAbort 与 mbIgnore，如图 10.3～图 10.7 所示。



图 10.3　MessageDlg 对话框 mbYesNoCancel 常数

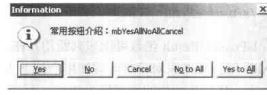


图 10.4　MessageDlg 对话框 mbYesAllNoAllCancel 常数



图 10.5　MessageDlg 对话框 mbOKCancel 常数

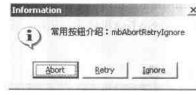


图 10.6　MessageDlg 对话框 mbAbortRetryIgnore 常数



图 10.7　MessageDlg 对话框 mbAbortIgnore 常数

上述几个常数分别代表各种常用的按钮，面对这些按钮，您同样可以使用范例 10-2 所介绍的方法获取用户所按下的按钮。当然，您也可以使用 IsAbortResult、IsAnAllResult、IsNegativeResult 或 IsPositiveResult 等函数来获取用户所按下的按钮。除此之外，您可以通过 StripAllFromResult 函数将众多可能的返回值加以分类，再做判断，分述如下：

- *IsAbortResult* 函数

IsAbortResult 函数可用来判断用户在对话框中是否按下 Abort 或 Cancel 等按钮结束对话框。假如用户按下 Abort 或 Cancel 等按钮时，IsAbortResult 函数的返回值为 True；反之，返回值为 False。

- *IsAnAllResult* 函数

IsAnAllResult 函数可用来判断用户在对话框中是否按下“All”、“Yes to All”或“No to All”等按钮结束对话框。假如用户按下“All”、“Yes to All”或“No to All”等按钮时，IsAnAllResult 函数的返回值为 True；反之，返回值为 False。

- *IsNegativeResult* 函数

IsNegativeResult 函数可用来判断用户在对话框中是否按下“No”或“No to All”等按钮结束对话框。假如用户按下“No”或“No to All”等按钮时，IsNegativeResult 函数的返回值为 True；反之，返回值为 False。

IsPositiveResult 函数

IsPositiveResult 函数可用来判断用户在对话框中是否按下“Ok”、“Yes”、“Yes to All”或“All”等按钮结束对话框。假如用户按下“Ok”、“Yes”、“Yes to All”或“All”等按钮时，IsPositiveResult 函数的返回值为 True；反之，返回值为 False。

StripAllFromResult 函数

StripAllFromResult 函数可以将含有“All”的按钮转换为单一的返回值，例如可以 mrAll 转为 mrOk；mrNoToAll 转为 mrNo；mrYesToAll 转为 mrYes。

范例 10-3

目的：本范例主要通过调用 MessageDlg 对话框来说明如何使用 IsAbortResult、IsAnAllResult、IsNegativeResult、IsPositiveResult 及 StripAllFromResult 等函数。

1 创建一个新的工程。

2 在 TForm 对象上加入一个 TButton 对象，并在此 TButton 对象的 onClick 事件处理过程中加入以下的代码。这段代码主要是通过 TmpResult 变量获取用户所按下的返回值，并使用 IsAbortResult、IsAnAllResult、IsNegativeResult 及 IsPositiveResult 等函数来判断用户所按下的按钮。最后，再通过 StripAllFromResult 函数将 TmpResult 变量转换为 mrOk、mrNo 或 mrYes 等返回值。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TmpResult : TModalResult;
  TmpResult2 : TModalResult;
begin
  TmpResult := MessageDlg('MessageDlg 对话框常用按钮',
                          mtInformation,mbYesAllNoAllCancel,0);
  //Abort or Cancel.
  if IsAbortResult(TmpResult) then
    Showmessage('转换前:IsAbortResult');
  //All, Yes to All, or No to All.
  if IsAnAllResult(TmpResult) then
    Showmessage('转换前:IsAnAllResult');
   //No or No to All.
  if IsNegativeResult(TmpResult) then
    Showmessage('转换前:IsNegativeResult');
  //Ok, Yes, Yes to All, or All.
  if IsPositiveResult(TmpResult) then
    Showmessage('转换前:IsPositiveResult');
  //可先进行转换
  TmpResult2 := StripAllFromResult(TmpResult);
```

```
    if TmpResult2 = mrOK then Showmessage('转换后:mrOK');
    if TmpResult2 = mrNo then Showmessage('转换后:mrNo');
    if TmpResult2 = mrYes then Showmessage('转换后:mrYes');
end;
```

③ 保存工程。

④ 按【F9】键运行工程。工程运行后，会出现“MessageDlg 对话框常用按钮”的对话框。您可以测试各种不同的按钮，并查看转换前后的不同。以图 10.8“YesToAll”为例。

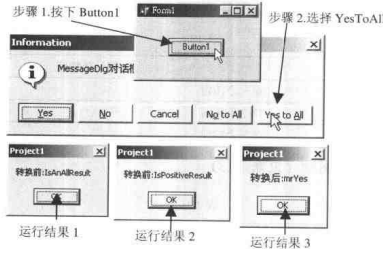


图 10.8　ModalResult 相关函数

10.3　MessageBox 对话框

MessageBox 对话框与 MessageDlg 对话框在功能上是相当类似的，两者同样可以掌握用户所按下的按钮。不同点在于 MessageBox 对话框可以进行自定标题文字。其函数声明如下：

```
function MessageBox(const Text, Capton: PChar,
                    Flags: Longint = MB_OK): Integer;
```

由 MessageBox 函数声明共有三个参数及一个返回值。第一个参数代表消息正文的内容、第二个参数代表对话框的标题文字、第三个参数是由一些常数所组合的，其默认值为 MB_OK。这些参数可以分为“出现的按钮组合”、“默认的按钮”、“显示的图标”及“运行模式”等四个类别来讨论，分述如下：

MessageBox 对话框按钮（见表 10.3）

表 10.3

常数值	出现的按钮
MB_OK	确定
MB_OkCancel	确定　取消

续上表

常数值	出现的按钮
MB_RetryCancel	重试(R) 取消
MB_AbortRetryIgnore	终止(A) 重试(R) 忽略(I)
MB_YesNo	是(Y) 否(N)
MB_YesNoCancel	是(Y) 否(N)

MessageBox 对话框默认按钮（见表 *10.4*）

表 10.4

常数值	默认的按钮
MB_DefButton1	第一个按钮
MB_DefButton2	第二个按钮
MB_DefButton3	第三个按钮

MessageBox 对话框图标（见表 *10.5*）

表 10.5

常数值	图标
MB_IconQuestion	?
MB_IconInformation	i
MB_IconAsterisk	i
MB_IconExcelamation	!
MB_IconStop	⊗
MB_IconHand	⊗

MessageBox 对话框模式（见表 *10.6*）

表 10.6

常数值	所属模式
MB_ApplModal	应用程序模式(在未结束对话框前也能切换到另一个应用程序)
MB_SystemModal	系统模式(您必须先结束对话框，才能做其他事)
MB_TaskModal	应用程序模式(在未结束对话框前也能切换到另一个应用程序)

MessageBox 对话框按钮返回值

表 10.7

各种按钮	返回值(常数)	返回值(整数)
对话框创建失败		0

续上表

各种按钮	返回值(常数)	返回值(整数)
确定	IdOk	1
取消	IdCancel	2
重试(R)	IdAbort	3
终止(A)	IdRetry	4
忽略(I)	IdIgnore	5
是(Y)	IdYes	6
否(N)	IdNo	7

范例 10-4

目的：本范例主要说明 MessageBox 对话框的使用方法。

1. 创建一个新的工程。
2. 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if Application.MessageBox('您要继续运行吗?','程序运行
            确认',MB_YesNo + MB_DEFBUTTON2+
            MB_IconStop+MB_ApplModal) <> IDYES then
    Halt;
end;
```

3. 保存工程。
4. 按【F9】键运行工程。工程运行后，会出现“您要继续运行吗?”的对话框，而且默认按钮为第二个按钮。假如您单击“是”按钮，则程序继续运行；反之，程序则立即结束，如图 10.9 所示。



图 10.9　MessageBox 对话框

注意

MessageBox 对话框事实上是 TApplication 对象的一个操作方法，因此当您调用 MessageBox 对话框时，必须先指定所属对象名称 Application。那什么又是 TApplication 对象呢？一个 Delphi 工程就对应于一个 TApplication 对象，Delphi 会自动为每一个工程产生一个 TApplication 对象的实例(Instance)，其名称为 Application。

10.4 InputBox 对话框

前面三种对话框用户都只能对现有的按钮做选择，接着为大家介绍一种能允许用户输入文本的对话框，我们称它为 InputBox 对话框。其函数声明如下：

```
function InputBox(const ACaption, APrompt, ADefault: string): string;
```

从这个函数声明中可以看出 InputBox 函数有三个参数，一个返回值。第一个参数是对话框的标题、第二个参数是对话框的提示文本、第三个参数则代表默认字符串，在用户单击“OK”按钮后，系统会将用户所输入的值返回给用户；反之，如果用户单击的是“Cancel”按钮，那么系统会将默认字符串返回给用户。

范例 10-5

目的：本范例主要说明 InputBox 对话框的使用方法。

步骤 1 创建一个新的工程。

步骤 2 在 TForm 对象上加入一个 TLabel 对象，其属性设置如表 10.8 所示。

表 10.8

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	欢迎光临
	Font.Size	12

步骤 3 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var Str:String;
begin
  //让用户输入姓名
  Str:=InputBox('姓名输入操作','请输入您的姓名',
                '樱木花道');
  //欢迎字符串
  Label1.Caption:=Str+','+Label1.Caption;
end;
```

步骤 4 保存工程。

步骤 5 按【F9】键运行工程。工程运行后，会出现“输入姓名操作”的对话框，如果您单击“OK”按钮，则返回您所输入的新字符串；否则，返回默认的文字，如图 10.10 所示。



图 10.10　InputBox 对话框

10.5　*InputQuery* 对话框

InputQuery 对话框也是个可输入文本的对话框，它与 InputBox 对话相当类似。但 InputQuery 对话框除了可接收用户所输入的字符串外，更可掌握用户所按下的按钮。其函数声明如下：

```
function InputQuery(const ACaption, APrompt: string;
                    var Value: string): Boolean;
```

InputQuery 函数同样有三个参数与一个返回值，第一个参数是对话框的标题、第二个参数是对话框的提示文本、第三个字符串是默认的字符串，它也是用来获取用户所输入字符串的变量。因为它是传址的变量传递(所以声明前面加了 Var)。此函数的返回值是一个布尔值(Boolean)，当返回值为 True，表示用户单击“OK”按钮；反之，则单击“Cancel”按钮。

范例 10-6

目的：本范例改写自范例 10-5，主要说明 InputQuery 对话框的使用方法。

步骤 1　创建一个新的工程。

步骤 2　在 TForm 对象上加入一个 TLabel 对象，其属性设置如表 10.9 所示。

表 10.9

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	欢迎光临
	Font.Size	12

步骤 3　在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Flag:Boolean;
  Str:String;
begin
  //让用户输入姓名
  Str:='樱木花道';
```

```
    Flag:=InputQuery('姓名输入操作','请输入您的姓名',Str);
    if Flag then
      //欢迎字符串
      Label1.Caption:=Str+','+Label1.Caption
    else
      Halt; //结束程序
end;
```

步骤 4 保存工程。

步骤 5 按【F9】键运行工程。工程运行后，会出现“输入姓名操作”的对话框，如果您单击“OK”按钮，则返回您所输入的新字符串，否则结束程序，如图 10.11 所示。

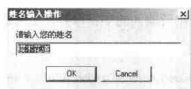


图 10.11 InputQuery 对话框

10.6 OpenDialog 与 SaveDialog 对话框

OpenDialog 与 SaveDialog 两个对话框是处理文件时经常使用的，目的就是让用户打开或保存指定的文件夹。由于这两个对话框的属性及操作方法完全相同，因此我们在此一起讨论。以下就其常用的属性及操作方法加以说明，如图 10.12 所示。



图 10.12 OpenDialog 与 SaveDialog 对话框

- *Title 属性*

Title 属性主要是决定对话框的标题文字，例如：“保存”、“打开”、“另存为”等。

- *InitialDir 属性*

InitialDir 属性主要是决定对话框的起始目录。

FileName 属性

FileName 属性代表着用户所选择的文件名称，或是对话框打开时的默认文件名称。

Filter 属性

Filter 属性主要是用来设置文件的过滤条件，如果您要编辑文件的过滤条件，只要在 Filter 属性字段点选，您就可以打开“Filter Editor”编辑器加以编辑，如图 10.13 所示。

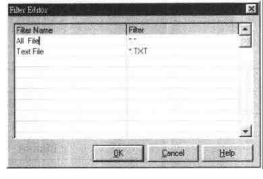


图 10.13　文件过滤条件编辑器

FilterIndex 属性

FilterIndex 属性是用来决定使用哪一个“文件过滤条件”，这里的文件过滤条件主要在 Filter 属性中设置。

DefaultExt 属性

DefaultExt 属性主要是用来指定默认的扩展名，当用户在输入的文件名中没有指定其扩展名，系统就会自动加入默认的扩展名。一般在使用时不需要加入“.”符号。

Options 属性

除了上述的属性外，在 OpenDialog 及 SaveDialog 对话框中尚有一些重要的属性，我们将它归类于 Options 属性。由于 Options 属性是一个复合属性，其细节属性通常只有 True 及 False 两种属性值，但这些细节属性并非经常被使用，一般来说您只要使用默认值就可以，如果您想再进一步的了解，请参考联机帮助说明。

OptionsEx 属性

OptionsEx 属性只包含 ofExNoPlacesBar 选项。当 ofExNoPlacesBar 为 True 时，表示此对话框不包含 Places Bar(如图 10.12 右所示)；反之，当 ofExNoPlacesBar 为 False 时，表示此对话框包含 Places Bar。其默认值为 False。

Files 属性

Files 属性通常用于 OpenDialog 对话框，当您选择多个文件时，系统会将这些文件名称

保存在这个 Files 属性，Files 属性继承于 TString 对象。通常必须配合 Options 的 OfAllowMultiSelect 属性值为 True 使用。

范例 10-7

目的：本范例上要说明 OpenDialog 及 SaveDialog 对话框 Files 属性的使用方法。

STEP 1 创建新的工程

STEP 2 在 TForm 对象上加入一个 TOpenDialog、一个 TListBox 及一个 TButton 对象，其属性设置如表 10.10 所示。

表 10.10

对象	属性名称	属性设置值
TOpenDialog	Name	OpenDialog1
TListBox	Name	ListBox1
TButton	Name	Button1
	Caption	Open File

STEP 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
    I:integer;
begin
   //设置 OpenDialog 属性
   OpenDialog1.Options := [ofAllowMultiSelect, ofFileMustExist];
   OpenDialog1.Filter := 'Text files (*.txt)|*.txt|All files (*.*) *.*';
   OpenDialog1.FilterIndex := 2;

   //打开文件
   if OpenDialog1.Execute then
      begin
         //清除 ListBox，再将新的文件加入
         ListBox1.Clear;
         with OpenDialog1.Files do
               for I := 0 to Count - 1 do
                    Listbox1.Items.add(Strings[I]);
      end;
end;
```

STEP 4 保存工程。

STEP 5 按【F9】键运行工程。工程运行后，会将您所打开的文件名称加到 TListBox 对象中，如图 10.14 所示。



图 10.14　OpenDialog 的 Files 属性

Execute 操作方法

我们通常使用 Execute 操作方法来打开 OpenDialog 或 SaveDialog 对话框，假如用户选择文件名称而且单击“打开文件”或“保存文件”等确定性质的按钮，则返回 Truc；反之，则返回 False。

范例 10-8

目的：本范例主要说明 OpenDialog 及 SaveDialog 对话框的使用方法。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象上加入一个 TMemo 对象、一个 OpenDialog 对话框、一个 SaveDialog 对话框及两个 TButton 对象，其属性设置如表 10.11 所示。

表 10.11

对象	属性名称	属性设置值
TMemo	Name	Memo1
TButton	Name	Button1
TButton	Name	Button2
TOpenDialog	Name	OpenDialog1
	InitialDir	C:\　　(不写代表目前的)
	Title	打开
	Filter	All File\|*.*\|Text File\|*.TXT
TSaveDialog	Name	SaveDialog1
	InitialDir	C:\　　(不写代表目前的)
	Title	保存
	DefaultExt	TXT
	Filter	All File\|*.*\|Text File\|*.TXT

步骤 3　在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  //打开文件
  if OpenDialog1.Execute then
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

步骤 4 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //保存文件
  if SaveDialog1.Execute then
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

步骤 5 保存工程。

步骤 6 按【F9】键运行工程。工程运行后，您可以打开及保存任何文本格式的文件数据，如图 10.15 所示。



图 10.15 OpenDialog 及 SaveDialog 对话框应用范例

10.7 OpenPictureDialog 与 SavePictureDialog 对话框

OpenPictureDialog 与 SavePictureDialog 两个对话框要算是 OpenDialog 及 SaveDialog 的应用特例，所以有很多的属性都是相当类似的，因此这里就不再赘述了。

范例 10-9

目的：本范例主要说明 OpenPictureDialog 及 SavePictureDialog 对话框的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TImage 对象(位于组件面板的 Additional 页标签)及两个 TButton 对象，其相关属性设置如表 10.12 所示。

表 10.12

对象	属性名称	属性设置值
TImage	Name	Image1
	AutoSize	True
TButton	Name	ButtonLoad
	Caption	Load
TButton	Name	ButtonSaveAs
	Caption	Save As

STEP 3 在 ButtonLoad 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonLoadClick(Sender: TObject);
begin
  //打开图片文件
  if OpenPictureDialog1.Execute then
Image1.Picture.LoadFromFile(OpenPictureDialog1. Filename);
end;
```

STEP 4 在 ButtonSaveAs 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonSaveAsClick(Sender: TObject);
begin
  //保存图片文件
  if SavePictureDialog1.Execute then
Image1.Picture.SaveToFile(SavePictureDialog1. Filename);
end;
```

STEP 5 保存工程。

STEP 6 按【F9】键运行工程。工程运行后您可以利用“Load”按钮打开图形文件，并且可以利用“Save As”按钮另存文件，如图 10.16 所示。



图 10.16　OpenPictureDialog 及 SavePictureDialog 应用范例

范例 10-10

目的：在范例 10-9 中一次只能打开一个指定的图片文件，似乎有些不方便，以下就示范如何配合文件处理相关组件来浏览图形文件。有关文件处理组件请参考第 11 章的详细说明。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TDriveComboBox 对象、一个 TDirectoryListBox 对象、一个 TFilterComboBox 对象、一个 TFileListBox 对象、六个 TLabel 对象、一个 TPanel 对象及一个 Timage 对象等对象，其相关属性设置如表 10.13 所示。

表 10.13

对象	属性名称	属性设置值
TDriveComboBox	Name	DriveComboBox1
	DirList	DirectoryListBox1
TDirectoryListBox	Name	DriectoryListBox1
	DirLabel	LabelPath
	FileList	FileListBox1
TFilterComboBox	Name	FilterComboBox
	FileList	FileListBox1
	Filter	(可复制自 TopenPictureDialog 对象的 Filter)
TFileListBox	Name	FileListBox1
TLabel	Name	Label1
	Caption	路径：
TLabel	Name	Label2
	Caption	图片高度：
TLabel	Name	Label3
	Caption	图片宽度：
TLabel	Name	LabelPath
TLabel	Name	LabelHeight
TLabel	Name	LabelWidth
TPanel	Name	Panel
	Caption	空白
TImage	Name	Image1(加在 Panel 对象上)
	Align	alClient

步骤 3 在 TFileListBox 对象的 onClick 事件处理过程中加入以下代码：

```
Procedure TForm1.FileListBox1Click(Sender: TObject);
begin
    //当用户有点选文件时，将这个图片文件加载
```

```
    Image1.Picture.LoadFromFile(FileListBox1.FileName);
    LabelHeight.Caption:=IntTostr(Image1.Picture.Height);
    LabelWidth.Caption:=IntTostr(Image1.Picture.Width);
end;
```

④ 保存工程。

按【F9】键运行工程。工程运行后，您可以利用文件处理组件找出指定的图形文件，再通过 TImage 对象将其显示。另外，我们也可以通过 TImage 对象的 Picture 属性获取该图片的长度与高度，如图 10.17 所示。



图 10.17　图形文件浏览应用范例

10.8　FontDialog 与 ColorDialog 对话框

FontDialog 与 ColorDialog 是文本处理常用的对话框。FontDialog 可以让用户按其需要来选择字体、字形样式、字体大小、字体效果及字体颜色等。而且用户可以立即从 FontDialog 所提供的范例字段得到所选择的设置结果，如图 10.18 所示。

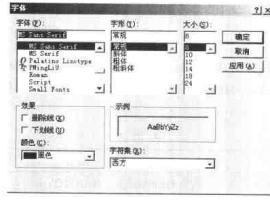


图 10.18　FontDialog 对话框

FontDialog 对话框是针对文本所设置的。而 ColorDialog 对话框所适用的对象包括所有对象的颜色相关属性。因此如果您只是想改变文本颜色事实上您也可以利用 ColorDialog 对话框来完成。在 Color 对话框中，用户可以按个人爱好从“基本颜色”或“自定义颜色”中选择所需要的颜色，如图 10.19 所示。

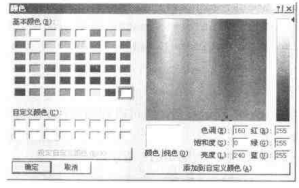


图 10.19 ColorDialog 对话框

10.8.1 FontDialog 常用属性与操作方法

以下就 FontDialog 对话框常用的属性及操作方法加以说明：

Device 属性

一般来说我们所指的字体都是指屏幕而言，但有些情况也会需要对打印机作字体的设置，Device 属性可以决定 FontDialog 对话框所设置字体的影响设备，Device 属性有三个可能的属性值，如表 10.14 所示。

表 10.14

Device 属性值	说明
fdScreen	影响设备为屏幕
fdPrinter	影响设备为打印机
fdBoth	影响设备为屏幕与打印机

Font 属性

Font 属性是用来记录用户在 FontDialog 对话框所做的相关选择，Font 属性是一个复合的属性，其包括 Charset、Color、Height、Name、Size、Style 等，请参考图 10.18，详细说明如表 10.15 所示。

表 10.15

Font 属性值	说明
Charset	所代表的字集，一般为 DEFAULT_CHARSET
Color	代表字体的颜色，其继承自 TColor 对象
Height	代表字体的高度
Name	代表字体名称
Size	代表字体的大小
Style	代表字体的样式，包括 fsBold(粗体),fsItalic(斜体), fsUnderline(下划线), fsStrikeOut(删除线)等

MaxFontSize 与 MinFontSize 属性

MaxFontSize 与 MinFontSize 属性是用来决定用户在 FonDialog 对话框中可以选择字体大小的最大值与最小值。必须配合 Options.fdLimitSize 属性值为 True 一起使用。

Options 属性

Options 属性是用来设置 FontDialog 对话框的详细选项，这些详细选项以有 True 与 False 两种可能值，表 10.16 将这些常用的选项加以说明。

表 10.16

Options 属性详细选项	说明
fdAnsiOnly	当属性值为 True 时，表示用户只能选取使用在 Windows 字集的字体
fdTrueTypeOnly	当属性值为 True 时，表示只会出现具有 TrueType 的字体
fdEffects	当属性值为 True 时，在 FontDialog 对话框中会出现“效果选项”，包括下划线、删除线及颜色等
fdFixedPitchOnly	当属性值为 True 时，在 FontDialog 对话框中只会出现固定宽度的字体名称供用户选择
fdForceFontExist	当属性值为 True 时，系统会自动检查用户所输入的字体名称是否存在，如图 10.20
fdNoFaceSel	当属性值为 True 时，FontDialog 对话框中的字体字段不会出现任何的字体名称
fdNoOEMFonts	当属性值为 True 时，FontDialog 对话框中只会出现非向量的字体
fdNoSimulations	当属性值为 True 时，FontDialog 对话框中只会出现非图形设备接口(GDI)的字体
fdNoSizeSel	当属性值为 True 时，FontDialog 对话框中的字号不会出现任何值
fdNoStyleSel	当属性值为 True 时，FontDialog 对话框中的字体样式不会出现任何值
fdNoVectorFonts	当属性值为 True 时，FontDialog 对话框中只会出现非向量的字体
fdShowHelp	当属性值为 True 时，FontDialog 对话框中会出现辅助说明按钮(Help)
fdWysiwyg	当属性值为 True 时，FontDialog 对话框中会出现同时可适用于屏幕与打印机的字体
fdLimitSize	当属性值为 True 时，MaxFontSize 与 MinFontSize 属性值才会有效
fdScalableOnly	当属性值为 True 时，FontDialog 对话框中只会出现能改变大小的字形
FdApplyButton	当属性值为 True 时，会出现“应用”的按钮，此按钮必须配合“onApply”事件一起使用



图 10.20 代表所选的字体不存在

Execute 操作方法

我们通常使用 Execute 操作方法来打开 FontDialog 对话框，假如用户单击“确定”按钮，则返回 True；反之，则返回 False。

OnApply 事件处理

当 FontDialog 对话框 Options 的详细属性 fdApplyButton 被设为 True 时，会出现“应用”按钮，而当用户单击此按钮时会触发 onApply 事件。因此简单地说，onApply 按钮可以让用户在关闭 FontDialog 对话框前就立即改变各项设置值。

范例 10-11

目的：本范例主要配合 TRichEdit 对象来说明 FontDialog 对话框的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TRichEdit、一个 TPopupMenu 及一个 FontDialog 对象，其相关属性设置如表 10.17 所示。

表 10.17

对象	属性名称	属性设置值
TRichEdit	Name	RichEdit1
	HideSelection	False
	Align	alClient
	PopupMenu	PopupMenu1
TPopupMenu	Name	PopupMenu1
	Items	FontMenuItem
FontMenuItem	Caption	字体
TFontDialog	Name	FontDialog1
	Options	[fdEffects,fdApplyButton]

步骤 3 在 TPopupMenu 的 FontMenuItem 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.FontMenuItemClick(Sender: TObject);
begin
  //假如用户有选取字符串，则将这些字符串的属性指定给
  //FontDialog
```

```
  FontDialog1.Font.Assign(RichEdit1.SelAttributes);
  if FontDialog1.Execute then
    begin
      //假如用户有选取字符串就改变它否则全体字符串都更改
      if RichEdit1.SelLength > 0 then
        RichEdit1.SelAttributes.Assign(FontDialog1.Font)
      else
        RichEdit1.DefAttributes.Assign(FontDialog1.Font);
    end;
end;
```

④ 在 TFontDialog 对象的 onApply 事件处理过程中加入以下代码：

```
procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  //假如用户有选取字符串就改变它否则全体字符串都更改
  if RichEdit1.SelLength > 0 then
    RichEdit1.SelAttributes.Assign(FontDialog1.Font)
  else
    RichEdit1.DefAttributes.Assign(FontDialog1.Font);
end;
```

⑤ 保存工程。

⑥ 按【F9】键运行工程。工程运行后，您可以选取字符串，并按下右键打开“FontDialog”对话框设置相关属性，如图 10.21 所示。



图 10.21　FontDialog 应用范例

10.8.2　ColorDialog 常用属性与操作方法

以下就 ColorDialog 对话框常用的属性及操作方法加以说明：

- *Color* 属性

Color 属性会记录用户所选择的颜色。

Options 属性

Options 属性是用来设置 ColorDialog 对话框的详细选项，这些详细选项只有 True 与 False 两种可能值，表 10.18 将这些常用的详细选项加以说明。

表 10.18

Options 属性详细选项	说明
cdFullOpen	当属性值为 True 时，表示 ColorDialog 对话框被打开时包含“自定义颜色”部分，请参考图 10.19
cdPreventFullOpen	当属性值为 True 时，表示 ColorDialog 对话框中的“规定自定义颜色”按钮呈无效状态。CdPreventFullOpen 选项必须配合 cdFullOpen 选项使用，当 cdFullOpen 选项为 True 时，“规定自定义颜色”按钮也会呈无效的状态
cdShowHelp	当属性值为 True 时，表示 ColorDialog 对话框中会出现“帮助”的说明按钮
cdSolidColor	当属性值为 True 时，表示用户在 ColorDialog 对话框中所选择的颜色会被转换为 Windows 中较接近的颜色
cdAnyColor	当属性值为 True 时，表示用户可以选择任何的颜色

Execute 操作方法

我们通常使用 Execute 操作方法来打开 ColorDialog 对话框，假如用户单击“确定”按钮，则返回 True；反之，则返回 False。

范例 10-12

目的：本范例主要说明 ColorDialog 对话框的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入两个 TLabel 对象、一个 TEdit 对象、一个 TShape 对象及一个 TButton 对象，其属性设置如表 10.19 所示。

表 10.19

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	颜色代号
TLabel	Name	Label2
	Caption	颜色结果
TEdit	Name	Edit1
TShape	Name	Shape1
TButton	Name	Button1
	Caption	Open

③ 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //设置联机帮助说明按钮，配合 TColorDialog 的
  //HelpContext 属性与 TForm 的 HelpFile 属性一起使用
  ColorDialog1.Options:=[cdShowHelp];

  if  ColorDialog1.Execute then
      begin
          //将用户选择的颜色转成字符串
          Edit1.Text:=ColorToString(ColorDialog1.Color);
           //指定颜色给 Shape1
          Shape1.Brush.color:=ColorDialog1.Color;
      end;
end;
```

④ 保存工程。

⑤ 按【F9】键功能运行工程。工程运行后，只要您单击“Open”按钮便会打开“ColorDialog”对话框，并将您所选择的颜色指定给 TShape 对象，如图 10.22 所示。

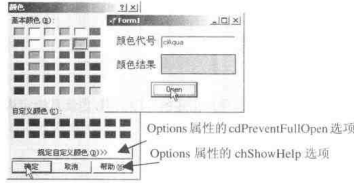


图 10.22　ColorDialog 应用范例

10.9　PrintDialog 与 PrintSetupDialog 对话框

一般的应用程序中几乎都免不了要有打印的功能，因此本小节要为大家介绍 PrintDialog 及 PrintSetupDialog 两个对话框。通过这两个对话框用户可以很容易的控制与设置打印机。

10.9.1　PrintDialog 对话框

PrintDialog 对话框是用来设置与打印有关的设置，如选择打印机、指定文字打印的范围、打印份数及打印至文件等设置，如图 10.23 所示。

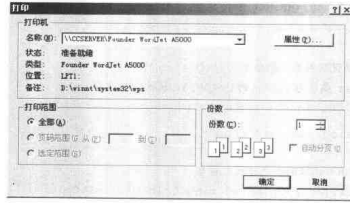


图 10.23 PrintDialog 对话框

以下就 PrintDialog 对话框的各项常用属性加以说明：

- *Collate 属性*

Collate 属性用来决定分页的 CheckBox 是否被选取，当 Collate 属性值为 True，分页的功能会被选取；反之，则不会被选取。

- *Copies 属性*

Copies 属性用来决定打印的份数。

- *FromPage 与 ToPage 属性*

FromPage 与 ToPage 属性用来记录用户指定的打印范围，也就是指定从哪一页(FromPage)打印到哪一页(ToPage)。

- *MaxPage 与 MinPage 属性*

MaxPage 与 MinPage 属性通常和 FormPage 与 ToPage 属性一起使用。MaxPage 属性用来设置 ToPage 属性的最大值。而 MinPage 属性用来设置 FromPage 的最小值。

- *PrintRange 属性*

在“打印范围”中有“全部”、“页数”及“选择的范围”等三个选项。这三个选项是由 PrintRange 属性来控制，表 10.20 是对 PrintRange 属性的值加以说明。

表 10.20

PrintRange 属性值	说明
prAllPages	当 PrintRange 属性值为 prAllPages 时，表示“全部”按钮被选取
PrPageNums	当 PrintRange 属性值为 prPageNums 时，表示“页数”按钮被选取。这项功能通常必须配合 Options 属性的 poPageNums 为 True 使用。由于这项功能可以让用户选择指定的页数，因此 MaxPage、MixPage、FromPage 及 ToPage 属性都必须正确的设置

续上表

PrintRange 属性值	说明
PrSelection	当 PrintRange 属性值为 prlSelection 时，表示“选择的范围”的按钮被选取。这项功能通常必须配合 Options 属性的 poSelection 为 True 使用

PrintToFile 属性

PrintToFile 属性用来改变或记录“打印到文件”的 CheckBox 按钮被选取状态。当 PrintToFile 属性值为 True 时，表示“打印到文件”的 CheckBox 按钮被选取；反之，则不被选取。如果 PrintToFile 属性值必须配合 Options 属性的 poPrintToFile 选项为 True 才有效。

Options 属性

Options 属性是用来设置 PrintDialog 对话框的详细选项，这些详细选项只有 True 与 False 两种可能值，表 10.21 将这些常用的详细选项加以说明。

表 10.21

Options 属性详细选项	说明
poPrintToFile	当 opPrintToFile 被设为 True 时，则 PrintDialog 对话框会出现“打印到文件”的复选框(CheckBox)，以供用户将数据打印到文件而非打印机
poPageNums	当 poPageNums 被设为 True 时，则“指定打印页数”的 RadionButton 成为有效
poSelection	当 poSelection 被设为 True 时，则“选择的范围”的 RadionButton 成为有效
poWarning	当 poWarning 被设为 True 时，假如指定的打印机尚未安装成功时，当用户单击“确定”按钮时会出现警告的消息
poHelp	当 poHelp 被设为 True 时，PrintDialog 对话框会出现帮助说明按钮
poDisablePrintToFile	当 poDisablePrintToFile 时，PrintDialog 对话框中的“打印到文件”复选框(CheckBox)会成为无效

Execute 操作方法

我们通常使用 Execute 操作方法来打开“PrintDialog”对话框，假如用户单击“确定”按钮，则返回 True；反之，则返回 False。

10.9.2　*PrintSetupDialog 对话框*

PrintSetupDialog 对话框主要是方便我们在程序里改变打印机设置。我们可以直接使用 Execute 操作方法来打开它，其打开方式为 PrintSetup.Execute。打开之后的画面如图 10.24 所示，但我们通常会利用 PrintDialog 对话框中的“属性”按钮来打开它。与打印机有关的设置被存在 win.ini 文件中。当然，这必须视您所使用的操作系统而定。其部分相关内容如下：

```
[Devices]
Epson EPL-5600=HPPCL5MS,LPT1:
[PrinterPorts]
```

Epson EPL-5600=HPPCL5MS,LPT1:,15,45

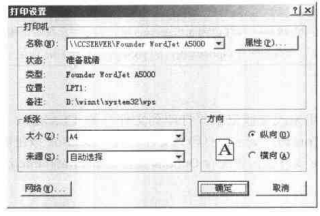


图 10.24 PrintSetUpDialog 对话框

范例 10-13

目的：本范例主要说明 PrintDialog 对话框的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TRichEdit 对象、一个 TOpenDialog 对象、一个 TPrintDialog 对象及两个 TButton 对象。其相关属性设置如表 10.22 所示。

表 10.22

对象	属性名称	属性设置值
TRichEdit	Name	RichEdit1
TOpenDialog	Name	OpenDialog1
TPrintDialog	Name	PrintDialog1
TButton	Name	ButtonOpen
	Caption	打开文件
TButton	Name	ButtonPrint
	Caption	打印

步骤 3 在 ButtonOpen 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
begin
  //您可以打开 Program Files\Borland\delphi6\demos\richedit\
  //overview.rtf
  if OpenDialog1.execute then
     RichEdit1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

步骤 4 在 ButtonPrint 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPrintClick(Sender: TObject);
begin
  //设置 PrintDialog 及打开 PrintDialog
  PrintDialog1.Options := [poSelection];
  if PrintDialog1.execute then
     RichEdit1.Print(Caption);
end;
```

5 保存工程。

6 按【F9】键运行工程。在工程运行后，您可以利用“打印”按钮打印您所指定的文字，如图 10.25 所示。



图 10.25　PrintDialog 应用范例

10.10　FindDialog 与 ReplaceDialog 对话框

在一般的文字处理中“查找”与“替换”是常见的功能，在 Delphi 6 中您可以通过 FindDialog 与 ReplaceDialog 两个对话框来完成。由于两者属性与操作方法相当类似，惟一的不同在于 ReplaceDialog 对话框多了 ReplaceText 属性与 onReplace 操作方法，如图 10.26 和图 10.27 所示。

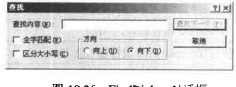


图 10.26　FindDialog 对话框

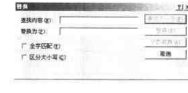


图 10.27　ReplaceDialog 对话框

以下就 FindDialog 与 ReplaceDialog 对话框常用的属性及操作方法加以说明：

- *FindText* 属性

FindText 属性是用来记录用户所要查找的特定字符串，适用于 FindDialog 与 ReplaceDialog 对话框。

- *ReplaceText* 属性

ReplaceText 属性是用来记录用户所要替代的特定字符串，适用于 ReplaceDialog 对话框。

Left 与 *Top* 属性

Left 属性是用来决定 FindDialog 与 ReplaceDialog 对话框出现时的 X 轴坐标，而 Top 属性是用来决定 FindDialog 与 ReplaceDialog 对话框出现时的 Y 轴坐标。

Position 属性

Position 属性是用来决定 FindDialog 与 ReplaceDialog 对话框出现时坐标，是 Left 与 Top 属性的集合体。

Options 属性

Options 属性是用来设置 PrintDialog 对话框的详细选项，这些详细选项只有 True 与 False 两种可能值，表 10.23 将这些常用的详细选项加以说明。

表 10.23

Options 属性详细选项	说明
frDown	当 frDown 选项为 True 时，表示查找方向为“向下”
frFindNext	当用户按下“找下一个”按钮时，frFindNext 属性被设为 True
frHideMatchCase	当 frHideMatchCase 选项为 True 时，表示对话框会隐藏“区分大小写”的选项
frHideWholeWord	当 frHideWholeWord 选项为 True 时，表示对话框会隐藏“全字匹配”的选项
frHideUpDown	当 frHideUpDown 选项为 True 时，表示对话框会隐藏查找“方向”的选项
frMatchCase	当 frMatchCase 选项为 True 时，表示对话框“区分大小写”的选项是被选取的
frDisableMatchCase	当 frDisableMatchCase 选项为 True 时，表示对话框“区分大小写”的选项是无效的(Disable)
frDisableUpDown	当 frDisableUp 选项为 True 时，表示对话框查找“方向”的选项是无效的(Disable)
frDisableWholeWord	当 frDisableWholeWord 选项为 True 时，表示对话框“全字匹配”的选项是无效的(Disable)
frReplace	当用户按下“替换”按钮时，frReplace 属性被设为 True
frReplaceAll	当用户按下“全部替换”按钮时，frReplaceAll 属性被设为 True
frWholeWord	当 frWholeWord 选项为 True 时，表示对话框“全字匹配”的选项是被选取的
frShowHelp	当 frShowHelp 选项为 True 时，表示对话框中会出现“帮助”的按钮

Execute 操作方法

我们通常使用 Execute 操作方法来打开 FindDialog 与 ReplaceDialog 对话框，假如用户单击“确定”按钮，则返回 True；反之，则返回 False。

OnFind 事件处理过程

当用户在 FindDialog 与 ReplaceDialog 对话框中按下“查找下一个”按钮时会触发 onFind 事件。

OnReplace 事件处理过程

当用户在 ReplaceDialog 对话框中按下“替换”或“全部替换”按钮都会触发 onReplace 事件。

范例 10-14

目的：本范例主要说明 FindDialog 与 ReplaceDialog 对话框的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TRichEdit 对象、一个 TOpenDialog 对象、一个 TFindDialog 对象、一个 TreplaceDialog 对象及三个 TButton 对象，其相关属性设置如表 10.24 所示。

表 10.24

对象	属性名称	属性设置值
TRichEdit	Name	RichEdit1
	HideSeletion	False
	ScrollBars	ssBath
TOpenDialog	Name	OpenDialog1
TFindDialog	Name	FindDialog1
TreplaceDialog	Name	ReplaceDialog1
TButton	Name	ButtonOpen
	Caption	打开文件
TButton	Name	ButtonFind
	Caption	查找
TButton	Name	ButtonReplace
	Caption	替换

STEP 3 在 ButtonOpen 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
begin
  //打开文件
  if OpenDialog1.execute then
    RichEdit1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

STEP 4 在 ButtonFind 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonFindClick(Sender: TObject);
begin
  //打开查找对话框
  FindDialog1.Execute;
```

```
end;
```

STEP 5 在 ButtonReplace 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonReplaceClick(Sender: TObject);
begin
  //打开替换对话框
  ReplaceDialog1.Execute;
end;
```

STEP 6 在 TFindDialog 对象的 onFind 事件处理过程中加入以下的代码：

```
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundPos,          //找到字符串的位置
  StartPos,          //开始查找的位置
  EndPos  :integer; //结束查找的位置
begin
  //利用判断 RichEdit 中是否有被选取的字符串来决定开始
  //查找的位置
  if RichEdit1.SelLength <> 0 then
    StartPos := RichEdit1.SelStart + RichEdit1.SelLength
  else
    //StartPos 为 0 表示从头开始查找
   StartPos := 0;
  //决定结束查找的位置
EndPos := Length(RichEdit1.Text) - StartPos;
  //利用 RichEdit1 中所提供的 FindText 来查找指定字符串的位置
  FoundPos := RichEdit1.FindText(FindDialog1.FindText,
                        StartPos, EndPos, [stMatchCase]);
  //当返回为-1 表示找不到字符串，反之则表示找到字符串
  if FoundPos <> -1 then
  begin
    //将找到的字符串加以反白
    RichEdit1.SetFocus;
    RichEdit1.SelStart := FoundPos;
    RichEdit1.SelLength := Length(FindDialog1.FindText);
  end;
end;
```

STEP 7 在 TReplaceDialog 对象的 onFind 事件处理过程中加入以下的代码：

```
procedure TForm1.ReplaceDialog1Find(Sender: TObject);
var
  FoundPos,          //找到字符串的位置
  StartPos,          //开始查找的位置
  EndPos  :integer; //结束查找的位置
begin
  //利用判断 RichEdit 中是否有被选取的字符串来决定开始
```

```
  //查找的位置
  if RichEdit1.SelLength <> 0 then
    StartPos := RichEdit1.SelStart + RichEdit1.SelLength
  else
    //StartPos 为 0 表示从头开始查找
   StartPos := 0;
  //决定结束查找的位置
EndPos := Length(RichEdit1.Text) - StartPos;
  //利用 RichEdit1 中所提供的 FindText 来查找指定字符串的位置
  FoundPos := RichEdit1.FindText(ReplaceDialog1.FindText,
                  StartPos, EndPos, [stMatchCase]);
  //当返回为-1 表示找不到字符串，反之则表示找到字符串
  if FoundPos <> -1 then
  begin
    //将找到的字符串加以反白
    RichEdit1.SetFocus;
    RichEdit1.SelStart := FoundPos;
    RichEdit1.SelLength := Length(ReplaceDialog1.FindText);
  end;
end;
```

8 在 TReplaceDialog 对象的 onReplace 事件处理过程中加入以下的代码：

```
procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
var
  FoundPos,          //找到字符串的位置
  StartPos,          //开始查找的位置
  EndPos  :integer; //结束查找的位置
begin
  //记录目前的位置
  StartPos:=RichEdit1.SelStart;
  //进行替换的工作
  RichEdit1.SelText := (ReplaceDialog1.ReplaceText);
  //重新设置反白区
  RichEdit1.SelStart:=StartPos;
RichEdit1.SelLength := Length(ReplaceDialog1.ReplaceText);
  with Sender as TReplaceDialog do
  if frReplaceAll in Options then
  begin
    Repeat
      //利用判断 RichEdit 中是否有被选取的字符串来决定开
      //始查找的位置
      if RichEdit1.SelLength <> 0 then
        StartPos := RichEdit1.SelStart + RichEdit1.SelLength
      Else
        //StartPos 为 0 表示从头开始查找
```

```
          StartPos := 0;
          //决定结束查找的位置
        EndPos := Length(RichEdit1.Text) - StartPos;
          //利用 RichEdit1 中所提供的 FindText 来查找指定
          //字符串的位置
        FoundPos := RichEdit1.FindText (ReplaceDialog1.FindText,    StartPos,
EndPos, [stMatchCase]);
          //当返回为-1 表示找不到字符串、反之则表示找到字符串
          if FoundPos <> -1 then
          begin
            //将找到的字符串加以反白
            RichEdit1.SetFocus;
            RichEdit1.SelStart := FoundPos;
          RichEdit1.SelLength := Length(ReplaceDialog1.FindText);
            //记录目前的位置
            StartPos:=RichEdit1.SelStart;
            //进行替换的工作
          RichEdit1.SelText := (ReplaceDialog1.ReplaceText);
            //重新设置反白区
            RichEdit1.SelStart:=StartPos;
            RichEdit1.SelLength := Length(ReplaceDialog1.
  ReplaceText);
        end;
      Until FoundPos = -1 ;
    end;
  end;
```

步骤 9 保存工程。

步骤 10 按【F9】功能键运行工程。工程运行后，您可以打开一个文件，并查找某个特定字符串，或以某特定字符串加以取之。在图 10.28 中示范如何将指定文件中的字符串“A”替换为字符串“Delphi 6 实务经典”。



图 10.28 ReplaceDialog 对话框

第11章

Delphi 6 常用对话框（二）

本章目标

本章将继续为您介绍 Delphi 6 其他常用的对话框，这些对话框包括 ShowMessagePos、ShowMessageFmt、MessageDlgPos、CreateMessageDialog、LoginDialog、LoginDialogEx、RemoteLoginDialog、SelectDirectory 及 PromptForFileName 等对话框。

本章主题

11.1 ShowMessagePos 对话框

11.2 ShowMessageFmt 对话框

11.3 MessageDlgPos 对话框

11.4 CreateMessageDialog 对话框

11.5 LoginDialog 对话框

11.6 LoginDialogEx 对话框

11.7 RemoteLoginDialog 对话框

11.8 SelectDirectory 对话框

11.9 PromptForFileName 对话框

11.1 ShowMessagePos 对话框

在第 10.1 节中笔者曾经介绍 ShowMessage 对话框，它是 Delphi 6 中最简单的消息对话框，相对的，其在功能上也有些限制。本小节将要介绍 ShowMessage 对话框的改良版，也就是 ShowMessagePos 对话框。ShowMessagePos 对话框与 ShowMessage 对话框最大的不同在于前者可以自行指定显示的位置，这个对话框的声明如下：

```
procedure ShowMessagePos(const Msg: string; X, Y: Integer);
```

ShowMessagePos 对话框的声明函数中包含三个参数，其中 Msg 参数代表显示的文字内容；X 与 Y 参数则代表对话框所要显示的坐标位置。以下就以实例来说明 ShowMessagePos 对话框。

范例 11-1

目的：本范例主要说明 ShowMessagePos 对话框的使用方法。

步骤 1 创建一个新的工程。

步骤 2 在 TForm 对象上加入两个 TLabel 对象、两个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.1 所示。

表 11.1

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	X
TLabel	Name	Label2
	Caption	Y
TEdit	Name	Edit1
	Text	(空白)
TEdit	Name	Edit1
	Text	(空白)
TButton	Name	Button1

步骤 3 为 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessagePos('ShowMessagePos对话框，显示坐标(' +
                 Edit1.Text + ',' + Edit2.Text + ')',
                 StrToInt(Edit1.Text),StrToInt(Edit2.Text));
end;
```

步骤 4　保存工程。

步骤 5　按【F9】功能键运行工程。工程运行后，您可以输入坐标位置再按下 Button1，ShowMessagePos 对话框即可出现于指定的位置，如图 11.1 所示。

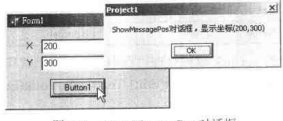


图 11.1　ShowMessagePos 对话框

11.2　ShowMessageFmt 对话框

在第 10.1 节中，笔者曾经介绍 ShowMessage 对话框，它是 Delphi 6 中最简单的消息对话框，相对的，其在功能上也有些限制。本小节将要介绍 ShowMessage 对话框的另一个改良版，也就是 ShowMessageFmt 对话框。ShowMessageFmt 对话框与 ShowMessage 对话框最大的不同在于前者可以针对所要显示的字符串内容进行格式化，这个对话框的声明如下：

```
procedure ShowMessageFmt(const Msg: string;
                         Params: array of const);
```

ShowMessageFmt 对话框的声明函数中包含两个参数，其中 Msg 参数代表显示的文字内容，但它是一个格化式的字符串；Params 参数则是以数组的方式代表 Msg 参数所需要的格化式参数内容。以下就以实例来说明 ShowMessageFmt 对话框。

范例 11-2

目的：本范例主要说明 ShowMessageFmt 对话框的使用方法。

步骤 1　创建一个新的工程。

步骤 2　在 TForm 对象上加入两个 TLabel 对象、两个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.2 所示。

表 11.2

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	年龄
TLabel	Name	Label2
	Caption	身高
TEdit	Name	Edit1
	Text	(空白)

续上表

对象	属性名称	属性设置值
TEdit	Name	Edit1
	Text	(空白)
TButton	Name	Button1

步骤 3 为 TButton 对象的 onClick 事件处理过程中加入以下的代码。其中%5d 与%-6d 代表两个被格式化的参数，而 StrToInt(Edit1.Text)与 StrToInt(Edit2.Text)则分别代表其所对应的参数值。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessageFmt('您的年龄是:%5d,身高为:%-6d 厘米',
      [StrToInt(Edit1.Text),StrToInt(Edit2.Text)]);
end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程。工程运行后，您可以输入坐标位置再按下 Button1，ShowMessageFmt 对话框即可出现被格式化的文字内容，如图 11.2 所示。

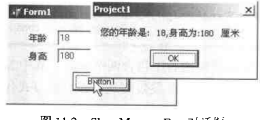


图 11.2 ShowMessageFmt 对话框

如果您仔细查看上述范例所显示的对话框，您可能会发现为何 18 与 180 两个数字排列的位置会不一样呢?那是因为%5d 与%-6d 两个格式化参数的缘故。其中%代表格式化的初始字符；5 与 6 分别代表所显示的字符个数；-代表左对齐，假如未指定则代表右对齐；d 则代表数字类型，这也就是为什么要使用 StrToInt 将 Edit1 与 Edit2 的内容值转为整数的道理了。除了 d 之外，Delphi 6 还提供了很多的数据类型供您选择。

- u 代表无正负号的十进制数(Unsigned decimal)。
- e 代表指数，因此其参数值必须为浮点数。
- f 代表固定格式的浮点数，例如“-ddd.ddd...”为参考格式。
- g 代表一般格式的浮点数。
- n 代表数字，其参数值必须为浮点数，例如“-d,ddd,ddd.ddd...”为参考格式。
- m 代表货币格式，其参数值必须为浮点数，显示的格式与 CurrencyString、CurrencyFormat、NegCurrFormat、ThousandSeparator、DecimalSeparatorpux CurrencyDecimals 等全局变量有关。
- p 代表指针格式(Pointer)，因此其参数值必须为指针变量。系统会以长度为 8 的字符串来显示 16 进制的指针变量。

- s 代表字符串，因此其参数值必须为字符(character)、字符串(string)或 PChar 等。
- x 代表 16 进制格式，因此其参数值必须为整数(integer)。

范例 11-3

目的：本范例说明 ShowMessageFmt 对话框的各种格式化参数。

STEP 1 创建一个新的工程。

STEP 2 在 TForm 对象上加入九个 TButton 对象，其相关属性设置如表 11.3 所示。

表 11.3

对象	属性名称	属性设置值
TButton	Name	ButtonU
	Caption	u 范例
TButton	Name	ButtonE
	Caption	e 范例
TButton	Name	ButtonF
	Caption	f 范例
TButton	Name	ButtonG
	Caption	g 范例
TButton	Name	ButtonN
	Caption	n 范例
TButton	Name	ButtonM
	Caption	m 范例
TButton	Name	ButtonP
	Caption	p 范例
TButton	Name	ButtonS
	Caption	s 范例
TButton	Name	ButtonX
	Caption	x 范例

STEP 3 在 ButtonU 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonUClick(Sender: TObject);
var
  d : Integer;
begin
  d := 12345;
  ShowMessageFmt('[u 范例]原始参数:12345,格式化后:%u',[d]);
end;
```

STEP 4 在 ButtonE 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonEClick(Sender: TObject);
var
  f : Real;
begin
  f := 0.12345;
  ShowMessageFmt('[e 范例]原始参数:0.12345,
格式化后:%e',[f]);
end;
```

5 在 ButtonF 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonFClick(Sender: TObject);
var
  f : Real;
begin
  f := -1234567.89;
  ShowMessageFmt('[f 范例]原始参数:-1234567.89,
格式化后:%f',[f]);
end;
```

6 在 ButtonG 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonGClick(Sender: TObject);
var
  f : Real;
begin
  f := -1234567.89;
  ShowMessageFmt('[g 范例]原始参数:-1234567.89,
格式化后:%g',[f]);
end;
```

7 在 ButtonN 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonNClick(Sender: TObject);
var
  f : Real;
begin
  f := -1234567.89;
  ShowMessageFmt('[n 范例]原始参数:-1234567.89,
格式化后:%n',[f]);
end;
```

8 在 ButtonM 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonMClick(Sender: TObject);
var
  f : Real;
begin
  f := -1234567.89;
```

```
  ShowMessageFmt('[m 范例]原始参数:-1234567.89,
格式化后:%m',[f]);
end;
```

STEP 9 在 ButtonP 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPClick(Sender: TObject);
type
  //定义一个指针类型
  PMyRecord = ^TMyRecord;
  TMyRecord = record
    Str : String;
  end;
var
   //声明一个指针变量
   MyRecordPointer: PMyRecord;
begin
  //创建指针变量，并分配内存
  New(MyRecordPointer);
  ShowMessageFmt('[p 范例]指针变量格式化后:%p',
[MyRecordPointer]);
  //释放内存
  FreeMem(MyRecordPointer);
end;
```

STEP 10 在 ButtonS 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonSClick(Sender: TObject);
var
  c : Char;
  s : string;
  p : Pchar;
begin
  c := 'A';
  s := 'My string';
  p := pchar('This is a PChar string');
  ShowMessageFmt('[s 范例]字符:%s,字符串:%s,PChar 字符串:%s',
[c,s,p]);
end;
```

STEP 11 在 ButtonX 对象 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonXClick(Sender: TObject);
var
   i : Integer;
begin
  i := 26;
```

```
    ShowMessageFmt('[x 范例]原始参数:26,格式化后:%x',[i]);
end;
```

步骤 12 保存工程。

步骤 13 按【F9】功能键运行工程。工程运行后，您可以查看各种不同格式化 ShowMessageFmt 对话框，如图 11.3～图 11.11 所示。



图 11.3 ShowMessageFmt 对话框 u 格式化范例



图 11.4 ShowMessageFmt 对话框 e 格式化范例



图 11.5 ShowMessageFmt 对话框 f 格式化范例



图 11.6 ShowMessageFmt 对话框 g 格式化范例

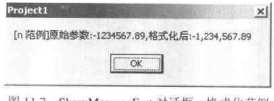


图 11.7 ShowMessageFmt 对话框 n 格式化范例

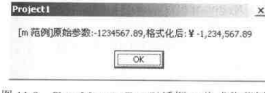


图 11.8 ShowMessageFmt 对话框 m 格式化范例

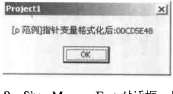


图 11.9 ShowMessageFmt 对话框 p 格式化范例

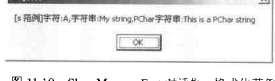


图 11.10 ShowMessageFmt 对话框 s 格式化范例



图 11.11 ShowMessageFmt 对话框 x 格式化范例

11.3 MessageDlgPos 对话框

在第 10.2 节中笔者曾经介绍 MessageDlg 对话框，本小节将为您介绍 MessageDlg 对话框的改良版，也就是 MessageDlgPos 对话框。MessageDlgPos 对话框与 MessageDlg 对话框最大的不同在于前者可以自行指定显示的位置，这个对话框的声明如下：

```
function MessageDlgPos(const Msg: string; DlgType: TMsgDlgType;
  Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer): Word;
```

MessageDlgPos 对话框的声明函数中包含六个参数，其中 Msg 参数代表显示的文字内容；DlgTypc 参数代表对话框的类型；Buttons 参数代表对话框中所要出现的按钮种类；HelpCtx 参数代表联机帮助说明；X 与 Y 参数则代表对话框所要显示的坐标位置。除了 X 与 Y 参数外，其余的参数都与 MessageDlg 对话框相同，笔者就不再赘述。以下就以实例来说明 MessageDlgPos 对话框。



11-4

目的：本范例主要说明 MessageDlgPos 对话框的使用方法。

Step 1 创建一个新的工程。

Step 2 在 TForm 对象上加入三个 TLabel 对象、二个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.4 所示。

表 11.4

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	X 坐标
TLabel	Name	Label2
	Caption	Y 坐标
TLabel	Name	Label2
	Caption	返回值
TEdit	Name	EditX
	Text	（空白）
TEdit	Name	EditY
	Text	（空白）
TEdit	Name	EditResult
	Text	（空白）
TButton	Name	Button1

Step 3 为 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ReturnValue : word;
begin
  ReturnValue := MessageDlgPos('请选择下项的按钮?',
```

```
        mtConfirmation, mbYesNoCancel, 0,
        StrToInt(EditX.Text), StrToInt(EditY.Text));
    case ReturnValue of
      mrCancel : EditResult.Text := 'Cancel';
      mrYes : EditResult.Text := 'Yes';
      mrNo : EditResult.Text := 'No';
    end;
  end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程。工程运行后，您可以输入坐标位置再按下 Button1，MessageDlgPos 对话框即可出现于指定的坐标位置，同时也可以获取用户所按下的按钮，如图 11.12 所示。

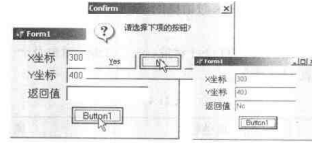


图 11.12 MessageDlgPos 对话框

11.4 CreateMessageDialog 对话框

CreateMessageDialog 对话框也可以算是 MessageDlg 对话框的一个特例。当某一种对话框可能会在工程中出现很多次时，您可以考虑创建一个 CreateMessageDialog 对话框变量，以供各个程序区块使用，这个对话框的声明如下：

```
function CreateMessageDialog(const Msg: string;
    DlgType: TMsgDlgType; Buttons: TMsgDlgButtons): TForm;
```

CreateMessageDialog 对话框的声明函数中包含三个参数，其中 Msg 参数代表显示的文字内容；Dlgtype 参数代表对话框的类型；Buttons 参数代表对话框中所要出现的按钮种类；其返回值为 TForm 对象类型。此函数的参数都与 MessageDlg 对话框相同，笔者就不再赘述。以下就以实例来说明 CreateMessageDialog 对话框。

范例 11-5

目的：本范例主要说明 CreateMessageDialog 对话框的使用方法。

步骤 1 创建一个新的工程。

(2) 在 TForm 对象上加入一个 TLabel 对象、一个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.5 所示。

表 11.5

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	返回值
TEdit	Name	EditResult
	Text	(空白)
TButton	Name	Button1

(3) 为 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Tmp : TForm;
  TmpResult : TModalResult;
begin
  //创建对话框
  Tmp := CreateMessageDialog( 这是由 CreateMessageDialog
         所产生的对话框',mtInformation,mbYesNoCancel);
  //显示对话框
  TmpResult := Tmp.ShowModal;
  //查看返回值
  case TmpResult of
    mrCancel : EditResult.Text := 'Cancel';
    mrYes : EditResult.Text :- 'Yes';
    mrNo : EditResult.Text :- 'No';
  end;
  //释放对话框
  Tmp.Free;
end;
```

(4) 保存工程。

(5) 按【F9】功能键运行工程。工程运行后再按下 Button1，CreateMessageDialog 对话框即被显示，同时也可以获取用户所按下的按钮，如图 11.13 所示。

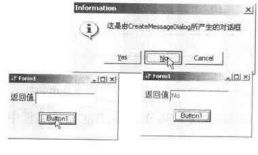


图 11.13　CreateMessageDialog 对话框

11.5 LoginDialog 对话框

编写数据库应用程序时，难免会需要用户输入登录的帐号与密码。在以前也许您和我一样都是使用 TForm 对象自行制作登录对话框。事实上，不用这么麻烦，因为 Delphi 6 提供 LoginDialog 对话框解决这方面的需求。这个对话框的声明如下：

```
function LoginDialog(const ADatabaseName: string;
                     var AUserName, APassword: string): Boolean;
```

LoginDialog 对话框的声明函数中包含三个参数，其中 ADatabaseName 参数代表数据库服务器名称；AUserName 参数代表登录数据库所使用的帐号；APassword 参数代表登录的密码；其返回值为 Boolean 类型。当用户单击“OK”按钮时，其返回值为 True；反之，则返回 False。以下就以实例来说明 LoginDialog 对话框。

范例 11-6

目的：本范例主要说明 LoginDialog 对话框的使用方法。

STEP 1 创建一个新的工程。

STEP 2 在 TForm 对象上加入三个 TLabel 对象、一个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.6 所示。

表 11.6

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	Database Server
TLabel	Name	Label2
	Caption	UserName
TLabel	Name	Label3
	Caption	Password
TEdit	Name	EditDatabaseName
	Text	(空白)
TEdit	Name	EditUserName
	Text	(空白)
TEdit	Name	EditPassword
	Text	(空白)
TButton	Name	Button1

STEP 3 因为 LoginDialog 对话框被定义在 DBLogDlg 单元文件中，因此我们必须在 uses 子句中引用此单元文件。

```
uses
```

```
  DBLogDlg;
```

④ 为 TButton 对象的 onClick 事件处理过程中加入以下的代码。这段代码中，笔者声明一个 DatabaseName 的常数。当然，您可以将其更改为指定的数据库名称。如果用户单击“OK”按钮，则系统会返回 True。因此我们就可以获取用户所输入的帐号与密码。

```
procedure TForm1.Button1Click(Sender: TObject);
const
  DatabaseName = 'carlos';
var
  UserName, Password: string;
begin
  if LoginDialog(DatabaseName,UserName,Password) then begin
    EditDatabaseName.Text := DatabaseName;
    EditUserName.Text := UserName;
    EditPassword.Text := Password;
  end;
end;
```

⑤ 保存工程。

⑥ 按【F9】功能键运行工程。工程运行后按下 Button1 时，系统会出现一个数据库的登录对话框。输入数据后，再单击“OK”按钮，程序即可获取帐号与密码，如图 11.14 所示。

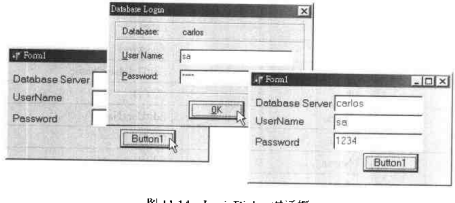


图 11.14　LoginDialog 对话框

11.6　LoginDialogEx 对话框

LoginDialog 对话框允许用户自行输入登录的帐号与密码，但有时侯您可能需要指定特定的帐号，也就是说帐号字段是只读的。那么这个时侯您可以使用 LoginDialogEx 对话框。这个对话框的声明如下：

```
function LoginDialogEx(const ADatabaseName: string; var
  AUserName, APassword: string; NameReadOnly: Boolean): Boolean;
```

LoginDialogEx 对话框的声明函数中包含四个参数，其中 ADatabaseName 参数代表数据库服务器名称；AUserName 参数代表登录数据库所使用的帐号；APassword 参数代表登录的密码；NameReadOnly 参数代表帐号字段是否为只读，当参数值为 True 表示只读；其返回值为 Boolean 类型。当用户单击“OK”按钮时，其返回值为 True；反之，则返回 False。以下就以实例来说明 LoginDialogEx 对话框。

范例 11-7

目的：本范例主要说明 LoginDialogEx 对话框的使用方法。

步骤 1 创建一个新的工程。

步骤 2 在 TForm 对象上加入一个 TLabel 对象、三个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.7 所示。

表 11.7

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	Database Server
TLabel	Name	Label2
	Caption	UserName
TLabel	Name	Label3
	Caption	Password
TEdit	Name	EditDatabaseName
	Text	(空白)
TEdit	Name	EditUserName
	Text	(空白)
TEdit	Name	EditPassword
	Text	(空白)
TButton	Name	Button1

步骤 3 因为 LoginDialogEx 对话框被定义在 DBLogDlg 单元文件中，因此我们必须在 uses 子句中引用此单元文件。

```
uses
  DBLogDlg;
```

步骤 4 为 TButton 对象的 onClick 事件处理过程中加入以下的代码。这段代码中，笔者声明一个 DatabaseName 的常数，及指定用户的帐号为 sa。当然，您可以将其更改为指定的数据库名称与帐号。如果用户单击“OK”按钮，则系统会返回 True。因此我

们就可以获取用户所输入的密码。

```
procedure TForm1.Button1Click(Sender: TObject);
const
  DatabaseName = 'carlos';
var
  UserName, Password: string;
begin
  UserName := 'sa';
  if LoginDialogEx(DatabaseName,UserName,Password,True) then begin
    EditDatabaseName.Text := DatabaseName;
    EditUserName.Text := UserName;
    EditPassword.Text := Password;
  end;
end;
```

5. 保存工程。

6. 按【F9】功能键运行工程。工程运行后按下 Button1 时，系统会出现一个数据库的登录对话框，但帐号是只读字段，输入密码后，再单击“OK”按钮，程序即可获取密码，如图 11.15 所示。

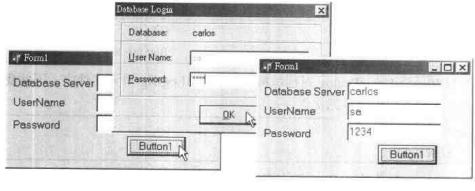


图 11.15　LoginDialogEx 对话框

11.7　*RemoteLoginDialog* 对话框

RemoteLoginDialog 对话框允许用户自行输入登录远程服务器的帐号与密码。这个对话框的声明如下：

```
function RemoteLoginDialog(var AUserName,
                           APassword: string): Boolean;
```

RemoteLoginDialog 对话框的声明函数中包含两个参数，其中 AUserName 参数代表登录远程服务器所使用的帐号；APassword 参数代表登录的密码；其返回值为 Boolean 类型。当

用户单击“OK”按钮时，其返回值为 True；反之，则返回 False。以下就以实例来说明 RemoteLoginDialog 对话框。

范例 11-8

目的：本范例主要说明 RemoteLoginDialog 对话框的使用方法。

STEP 1 创建一个新的工程。

STEP 2 在 TForm 对象上加入三个 TLabel 对象、三个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 11.8 所示。

表 11.8

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	UserName
TLabel	Name	Label2
	Caption	Password
TEdit	Name	EditUserName
	Text	(空白)
TEdit	Name	EditPassword
	Text	(空白)
TButton	Name	Button1

STEP 3 因为 RemoteLoginDialog 对话框被定义在 DBLogDlg 单元文件中，因此我们必须在 uses 子句中引用此单元文件。

```
uses
  DBLogDlg;
```

STEP 4 为 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  UserName, Password: string;
begin
  if RemoteLoginDialog(UserName,Password) then begin
    EditUserName.Text := UserName;
    EditPassword.Text := Password;
  end;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程。工程运行后按下 Button1 时，系统会出现一个远程服务器的登录对话框。输入数据后，再单击“OK”按钮，程序即可获取登录的帐号与密码，如图 11.16 所示。

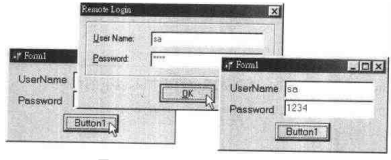


图 11.16　RemoteLoginDialog 对话框

11.8　SelectDirectory 对话框

在很多情况下，您的程序可能会需要用户选择或输入一个指定的目录作为保存或读取的目录。Delphi 6 提供 SelectDirectory 对话框协助您完成这种需求。这个对话框有两种的声明方式，分述如下：

```
function SelectDirectory(const Caption: string;
                         const Root: WideString; out Directory: string): Boolean;
```

SelectDirectory 对话框的第一种声明方式中包含三个参数，其中 Caption 参数代表对话框的标题文字；Root 参数代表指定的根目录名称；Directory 参数代表用户所指定的目录名称；其返回值为 Boolean 类别。当用户单击“OK”按钮时，其返回值为 True；反之，则返回 False。这里您必须特别注意 Root 参数与 Directory 参数不可以被指定来自相同的变量名称。以下就以实例说明 SelectDirectory 对话框。

范例 11-9

目的：本范例主要说明 SelectDirectory 对话框的使用方法。

1. 创建一个新的工程。
2. 在 TForm 对象上加入一个 TLabel 对象、一个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 10.9 所示。

表 10.9

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	目录名称
TEdit	Name	Edit1
	Text	(空白)
TButton	Name	Button1

3. 因为 SelectDirectory 对话框被定义在 FileCtrl 单元文件中，因此我们必须在 uses 子

句中引用此单元文件。

```
uses
  FileCtrl;
```

STEP 4 为 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Title, Root, Dir: string;
begin
  Title := 'Select Directory Function Testing 1';
  //注意 Root 参数，空值代表为最上层的根目录
  Root := 'F:\Delphi 6 实务经典\ ';
  if SelectDirectory(Title,Root,Dir) then Edit1.Text := Dir;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程。工程运行后您会发现用户只能在默认的目录或磁盘下选择指定目录，如图 11.17 所示。

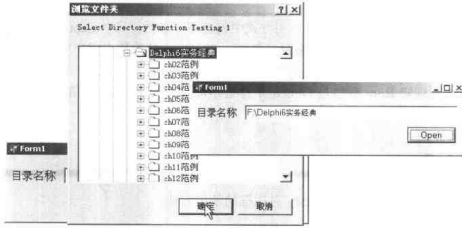


图 11.17 SelectDirectory 对话框(一)

SelectDirectory 对话框的另一个声明方式如下：

```
function SelectDirectory(var Directory: string;
                         Options: TSelectDirOpts; HelpCtx: Longint): Boolean;
```

SelectDirectory 对话框的第二种声明方式中包含三个参数，其中 Directory 参数代表出现在对话框的指定目录名称；Options 参数代表开启对话框时的选项设置，例如 Options 参数为空值时，代表用户只能选择已经存在的目录。假如不为空值，就必须视其参数值而定；HelpCtx 参数代表说明文件；其返回值为 Boolean 类型。当用户按下 OK 按钮时，其返回值为 True；反之，则返回 False。以下就以实例来说明 SelectDirectory 对话框。

范例 11-10

目的：本范例主要说明 SelectDirectory 对话框的另一个使用方法。本范例主要修改自范例 10-23。因此相同的部分就不再赘述。以下仅修改 Button1 的 onClick 事件处理过程内容。

步骤 1 为 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Dir: string;
begin
  //uses FileCtrl;
  //Dir 为空白代表当前的目录
  Dir := 'C:\';
  if SelectDirectory(Dir, [sdAllowCreate, sdPerformCreate,
                                                sdPrompt],0) then
    Edit1.Text := Dir;
end;
```

步骤 2 保存工程。

步骤 3 按【F9】功能键运行工程。工程运行后您可以选择不同的磁盘驱动器、目录、查看目录中的文件，甚至可以创建新的目录名称。以下图为例，笔者在“C:\temp”目录下创建一个名称为“56789”的子目录。

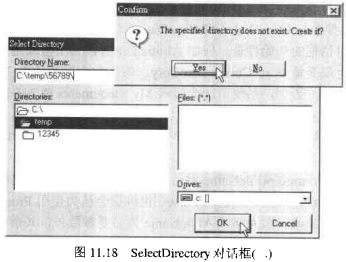


图 11.18　SelectDirectory 对话框(.)

注意　为什么这个 Select Directory 对话框可以有创建子目录的功能呢？那是因为 Options 参数的设置值。Options 参数包含 sdAllowCreate、sdPerformCreate 及 sdPrompt 等值，分述如下：sdAllowCreate 代表允许用户创建子目录；sdPerformCreate 代表运行创建子目录的功能，通常与 sdAllowCreate 参数一起使用；sdPrompt 参数代表询问用户是否创建子目录。

11.9 PromptForFileName 对话框

PromptForFileName 对话框也可视为 OpenDialog 与 SaveDialog 对话框的结合体。也就是说，PromptForFileName 对话框同时可以当做 OpenDialog 与 SaveDialog 对话框。这个对话框的声明如下。

```
function PromptForFileName(var AFileName: string;
        const AFilter: string = ''; const ADefaultExt: string = '';
        const ATitle: string = ''; const AInitialDir: string = '';
        SaveDialog: Boolean = False): Boolean;
```

PromptForFileName 对话框的声明函数中包含 AFileName、AFilter、ADefaultExt、ATitle、AInitialDir 及 SaveDialog 等六个参数，分述如下：

- AFileName 参数与 OpenDialog 和 SaveDialog 对话框的 FileName 属性相同，其代表用户所选择的文件名称，或是对话框打开时的默认文件名称。
- AFilter 参数与 OpenDialog 和 SaveDialog 对话框的 Filter 属性相同，其用来设置文件的过滤条件。有关过滤条件的格式请参考第 11.6 节的详细说明。
- ADefaultExt 参数与 OpenDialog 和 SaveDialog 对话框的 DefaultExt 属性相同，其主要是用来指定默认的扩展名。假如您未指定 ADefaultExt 参数值，系统默认的扩展名为*.*。
- ATitle 参数与 OpenDialog 和 SaveDialog 对话框的 Title 属性相同，其主要是用来决定对话框的标题文字。假如您未指定 ATitle 参数值，系统会根据对话框的类型自行决定对应的标题文字。
- AInitialDir 参数与 OpenDialog 和 SaveDialog 对话框的 InitialDir 属性相同，其主要是用来决定对话框的初始目录。假如 AInitialDir 参数为空值或是目录不存在时，系统会先判断全局变量 ForceCurrentDirectory。假如此全局变量为 True 时系统会显示目前的目录位置，否则，系统可能会显示 My Documents 或 NT 的工作目录，这完全决定于不同版本的操作系统。
- SaveDialog 参数用来决定对话框的种类，当参数值为 True 表示此为 SaveDialog 对话框；反之，则为 OpenDialog 对话框。其默认值为 False。

除了六个参数外，PromptForFileName 对话框的返回值为 Boolean 类型。当用户单击“OK”按钮时，其返回值为 True；反之，则返回 False。也许您会认为使用 PromptForFileName 对话框很麻烦，事实上并非如此。因为除了 AFileName 为必要参数外，其余都是可以省略的。以下就以实例来说明 PromptForFileName 对话框。

范例 11-11

目的： 本范例主要说明 PromptForFileName 对话框的使用方法。

STEP 1 创建一个新的工程。

STEP 2 在 TForm 对象上加入两个 TLabel 对象、两个 TRadioButton 对象、一个 TEdit 对象

及一个 TButton 对象，其相关属性设置如表 11.10 所示。

表 11.10

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	对话框类型
TLabel	Name	Label2
	Caption	文件名称
TRadioButton	Name	RadioButton1
	Caption	Save Dialog
TRadioButton	Name	RadioButton2
	Caption	Open Dialog
TEdit	Name	Edit1
	Text	(空白)
TButton	Name	Button1

步骤 3　因为 PromptForFileName 对话框被定义在 Dialogs 单元文件中，因此我们必须在 uses 子句中引用此单元文件。

```
uses
  Dialogs;
```

步骤 4　为 TButton 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AFileName, AFilter, ADefaultExt, ATitle, AInitialDir : String;
  SaveDialog : Boolean;
begin
  AFileName := 'Default';
  AFilter := 'Text files (*.txt)|*.TXT|All files (*.*)|*.*';
  ADefaultExt := '*.TXT';
  ATitle := '';
  AInitialDir := 'F:\Delphi 6 实务经典\ch11 范例\';
  SaveDialog := RadioButton1.Checked;
  if PromptForFileName(AFileName, AFilter, ADefaultExt,
              ATitle, AInitialDir, SaveDialog) then
    Edit1.Text := AFileName;
end;
```

步骤 5　保存工程。

步骤 6　按【F9】功能键运行工程。工程运行后您可以选择对话框类型并按下 Button1，此时系统会显示指定的对话框，这里以 OpenDialog 对话框为例，如图 11.19 所示。



图 11.19　PromptForFileName 对话框

第 12 章

Delphi 6 文件目录管理对象

本章目标

本章将为各位介绍 Delphi 6 的文件目录管理对象，包含 TDirveComboBox 对象、TDirectoryListBox 对象、TFileListBox 对象、TFilterComboBox 对象、TShellComboBox 对象、TShellTreeView 对象、TShellListView 对象及 TShellChangeNotifier 对象等。

本章主题

12.1 TdirveComboBox 对象

12.2 TDirectoryListBox 对象

12.3 TFileListBox 对象

12.4 TFilterComboBox 对象

12.5 TShellComboBox 对象

12.6 TShellTreeView 对象

12.7 TShellListView 对象

12.8 TShellChangeNotifier 对象

12.1 TDirveComboBox 对象

TDriveComboBox 对象提供一个驱动器的下拉式菜单，菜单中的选项包括目前系统内的所有驱动器，TDriveComboBox 对象位于组件面板的 Win31 页标签中。以下就 TDriveComboBox 对象中有几个较为常用的属性及事件加以介绍，分述如下：

TextCase 属性

TextCase 属性用来决定 TDriveComboBox 对象显示驱动器名称的大小写。当 TextCase 属性值为 tcUpperCase 时，表示显示的名称为大写；当 TextCase 属性值为 tcLowerCase 时，表示显示名称为小写。其默认值为 tcLowerCase。

Drive 属性

Drive 属性记录着用户所选择的驱动器名称。

DirList 属性

DirList 属性可以连接到一个 TDirectoryListBox 对象。TDirectoryListBox 对象会以树状结构的方式显示指定驱动器的所有目录名称。有关 TDirectoryListBox 对象请参考第 12.2 节的详细说明。

onChange 事件

当用户更改已选择的驱动器名称时，系统会触发 onChange 事件。

12.2 TDirectoryListBox 对象

TDirectoryListBox 对象会以树状结构的方式显示指定磁盘下的所有目录名称，TDirectoryListBox 对象位于组件面板的 Win31 页标签中。以下就 TDirectoryListBox 对象中有几个较为常用的属性、操作方法及事件处理加以介绍，分述如下：

Directory 属性

Driectory 属性记录着用户所选择的目录名称。

Drive 属性

Drive 属性记录着 TDirectoryListBox 对象目前所指定的驱动器名称。通常，我们会在 TDriveComboBox 对象的 onChange 事件处理过程中，将 TDirectoryListBox 对象的 Drive 属性值设置为 TDriveComboBox 对象的 Drive 属性值，如此一来，当 TDriveComboBox 对象改变驱动器位置时，TDirectoryListBox 对象也会自动更改其内容。另外有一个简单的方法可以达到这个功能，那就是将 TDriveComboBox 对象的 DirList 属性指定为 TDirectoryListBox 对象。

DirLabel 属性

DirLabel 属性可连接到指定的 TLabel 对象，该 TLabel 对象即可自动显示用户所选择的目录名称。

FileList 属性

FileList 属性可连接到指定的 TFileListBox 对象，通过 TFileListBox 对象显示指定目录下的所有文件名称。有关 TFileListBox 对象请参考第 12.3 节的详细说明。

PreserveCase 属性

PreserveCase 属性用来判断文件与目录名称是否保留文件系统原先所记录的字母大小写。当 PreserveCase 属性值为 True，表示保留；反之，则代表不保留。PreserveCase 属性是一个只读的属性，您只能读取，而无法更改。

CaseSensitive 属性

CaseSensitive 属性可用来判断目前的文件系统是否为 Case-Sensitive(有字母大小写之分)的系统。当 CaseSensitive 属性值为 True，表示目前的文件系统有字母大小写之分；反之，则代表没有大小写之分。

范例 12-1

目的：本范例主要说明 TDirectoryListBox 对象与 TDriveComboBox 对象的使用方法。

Step 1 创建新的工程。

Step 2 在 TForm 对象上加上一个 TDriveComboBox 对象、一个 TDirectoryListBox 对象及四个 TLabel 对象，其相关属性设置如表 12.1 所示。

表 12.1

对象	属性名称	属性设置值
TDriveComboBox	Name	DriveComboBox1
	DirList	DirectoryListBox1
TDirectoryListBox	Name	DirectoryListBox1
	DirLabel	LabelDir1
TLabel	Name	Label1
	Caption	目录名称(1)：
TLabel	Name	LabelDir1
TLabel	Name	Label2
	Caption	目录名称(2)：
TLabel	Name	LabelDir2

Step 3 在 DirectoryListBox1 对象的 onMouseUp 事件处理过程中加入以下代码，以获取 DirectoryListBox1 对象目前所指定的目录名称。

```
procedure TForm1.DirectoryListBox1MouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

```
begin
   Label2Dir2.Caption := DirectoryListBox1.Directory;
end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程，工程运行后您可以选择指定的驱动器或目录名称，系统自动显示目前所在的目录名称，如图 12.1 所示。



图 12.1 TDriveComboBox 与 TDirectoryListBox 对象的应用范例

注意 您可能会觉得很奇怪，为什么图 12.1 中“目录名称(1)”与“目录名称(2)”所显示的内容不一样呢?因为“目录名称(1)”是通过 TDirectoryListBox 对象的 DirLabel 属性获取目前的目录名称，因此如果目录名称太长，系统会自动加以简化。而“目录名称(2)”是通过 TDirectoryListBox 对象 Directory 属性获取目前的目录名称，因此它是完整的目录名称。

- *OpenCurrent* 操作方法

当您在 TDirectoryListBox 对象上双击鼠标时，系统会自动打开目前所指定的目录。当然，您也可以通过调用 TDirectoryListBox 对象的 OpenCurrent 操作方法来达到这个功能。

- *GetItemPath* 操作方法

GetItemPath 操作方法允许您通过选项索引的方式获取 TDirectoryListBox 对象上的目录名称，索引值由 0 算起。

- *onChange* 事件

当用户更改已选择的目录名称时，系统会触发 onChange 事件。

范例 12-2

目的：本范例主要说明 TDirectoryListBox 对象 OpenCurrent 操作方法的使用方法。

步骤 1 打开范例 12-1。

步骤 2 在 DirectoryListBox1 对象的 onClick 事件处理过程中加入以下的代码，以打开目前

所指定的目录。

```
procedure TForm1.DirectoryListBox1Click(Sender: TObject);
begin
  DirectoryListBox1.OpenCurrent;
end;
```

3) 保存并运行工程。工程运行后，当鼠标左键点选 DirectoryListBox1 对象时，系统会自动的打开指定的目录，就好像您使用双击鼠标一样的功能。

12.3　TFileListBox 对象

TDirectoryListBox 对象会以树状结构的方式显示指定磁盘下的所有目录名称，而 TFileListBox 对象则是显示指定目录下所有的文件名称。TFileListBox 对象位于组件面板的 Win31 页标签中。以下就 TFileListBox 对象中有几个较为常用的属性、操作方法及事件处理加以介绍，分述如下：

Directory 属性

Driectory 属性记录着目前所指定的目录名称，此属性值可以通过 TDirectoryListBox 对象的 Driectory 属性来获取。通常，我们会设置 TDirectoryListBox 对象的 FileList 属性为 TFileListBox 对象，以便系统自动显示与更新 TFileListBox 对象的内容值。

Drive 属性

Drive 属性代表目前所指定的驱动器名称，此属性值可以通过 TDirectoryListBox 对象或 TDriveComboBox 对象的 Drive 属性来获取驱动器名称。

FileType 属性

FileType 属性可决定 TFileListBox 对象所要显示的文件属性。其包括：ftReadOnly、ftHidden、ftSystem、ftVolumeID、ftDirectory、ftArchive 与 ftNormal 等。ftReadOnly 代表显示只读文件；ftHidden 代表显示隐藏文件；ftSystem 代表显示系统文件；ftVolumeID 代表显示卷册(Volume) ID 名称；ftDirectory 代表显示目录文件；ftArchive 代表显示保存文件；ftNormal 则代表显示一般文件。其默认值只包含一般文件。

Mask 属性

FileType 属性是由文件的类型来过滤所要显示的文件或目录，而 Mask 属性则是由文件或目录名称来过滤。Mask 属性值的设置方式，请参考第 10.6 节的 OpenDialog 对话框与 SaveDialog 对话框的 Filter 属性设置说明。其默认值为*.*，代表显示所有的文件。

MultiSelect 属性

MultiSelect 属性用来决定此 TFileListBox 对象是否允许用户同时选取多个数据项。当 MultiSelect 属性值为 True，表示此 TFileListBox 对象允许用户同时选取多个数据项，否则只

能选取一个数据项，其默认值为 False。有关 MultiSelect 属性的详细说明请参考第 8.3 节的范例说明。

ShowGlyphs 属性

ShowGlyphs 属性用来决定此 TFileListBox 对象是否在每一个数据项前面显示小图标。当 ShowGlyphs 属性值为 True，表示此 TFileListBox 对象会在每一个数据项前面显示小图标；反之，则不显示。

范例 12-3

目的： 本范例主要说明 TFileListBox 对象的使用方法。

(1) 打开范例 12-2。

(2) 在 Form 对象上加入一个 TFileListBox 对象与一个 TEdit 对象，其相关属性设置如表 12.2 所示。

表 12.2

对象	属性名称	属性设置值
TFileListBox	Name	FileListBox1
	FileEdit	Edit1
	ShowGlyphs	True
TEdit	Name	Edit1

(3) 将 DirectoryListBox1 对象的 FileList 属性设为 FileListBox1 对象。

(4) 保存并运行工程。工程运行后，当鼠标在 DirectoryListBox1 对象上任意点选指定目录时，系统会自动将此目录下的文件显示在 TFileListBox 对象上，如图 12.2 所示。



图 12.2 TFileListBox 对象的应用范例

ApplyFilePath 操作方法

ApplyFilePath 操作方法可以将指定的完整路径与文件名称分别应用到 TDriveComboBox、TDirectoryListBox 及 TFileListBox 等对象的相关属性值。

onChange 事件

当用户更改已选择的文件名称时，系统会触发 onChange 事件。

范例 12-4

目的：本范例主要说明如何使用 TFileListBox 对象的 ApplyFilePath 操作方法。

1. 打开范例 12-3。
2. 在 TForm 对象上加入一个 TEdit 对象与一个 TButton 对象，其相关属性设置如表 12.3 所示。

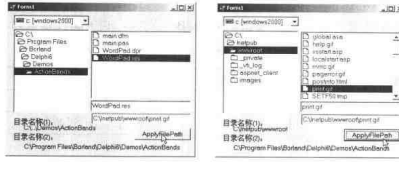
表 12.3

对象	属性名称	属性设置值
TEdit	Name	Edit2
TButton	Name	Button1
	Caption	ApplyFilePath

3. 将 Button1 对象的 onClick 事件处理程序中加入以下的代码，这段代码主要目的是将指定的路径应用到 TDriveComboBox、TDirectoryListBox 及 TFileListBox 等对象的相关属性值。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit2.Text);
end;
```

4. 保存并运行工程。工程运行后，您可以在 Edit2 对象上输入指定的路径与文件名称。再按下 Button1 按钮，此时系统会自动将此路径与文件名称应用到 TDriveComboBox、TDirectoryListBox 及 TFileListBox 等对象的相关属性值。假如您输入一个无效的路径或文件名称，系统并不会更改相关对象的原有属性设置，如图 12.3 所示。



应用前　　应用后

图 12.3　ApplyFilePath 操作方法应用范例

12.4 TFilterComboBox 对象

上一小节中，我们介绍了 TFileListBox 对象的 Mask 属性，它可以设置文件或目录名称的过滤条件，以决定 TFileListBox 对象所要显示的内容。为了使用上的方便，Delphi 另外提供 TFilterComboBox 对象，位于组件面板的 Win31 页标签中。您可以事先设置过滤的条件，以供用户在运行时期选用。以下就 TFilterComboBox 对象中几个较为常用的属性加以介绍，分述如下：

- *FileList 属性*

FileList 属性可设置过滤的对象，必须指定一个 TFileListBox 对象。

- *Filter 属性*

Filter 属性主要是用来设置文件的过滤条件，如果您要编辑文件的过滤条件，只要在 Filter 属性字段点选，就可以打开“Filter Editor”编辑器进行编辑，如图 12.4 所示。



图 12.4 Filter Editor 编辑器

范例 12-5

目的：本范例主要说明如何使用 TFilterComboBox 对象。

STEP 1 打开范例 12-4。

STEP 2 在 TForm 对象上加入一个 TFilterComboBox 对象，相关属性设置如表 12.4 所示。

表 12.4

对象	属性名称	属性设置值
TFilterComboBox	Name	FilterComboBox1
	Filter	再加入一个 *.txt 过滤条件
	FileList	FileListBox1

STEP 3 保存并运行工程。工程运行后，您可以选择 TFilterComboBox 对象事先设置的过滤条件以决定 FileListBox1 所要显示的内容，如图 12.5 所示。

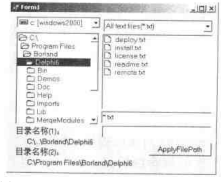


图 12.5　TFilterComboBox 对象应用范例



假如您结合 TFileListBox 对象与 TFilterComboBox 对象一起使用，那么 TFileListBox 对象的 Mask 属性将被 TFilterComboBox 对象的 Filter 属性替换。也就是说，如果您同时设置 TFileListBox 对象的 Mask 属性值与 TFilterComboBox 对象的 Filter 属性值时，系统将以 TFilterComboBox 对象的 Filter 属性为过滤条件。

12.5　TShellComboBox 对象

前面四个小节中所介绍的文件目录管理对象都位于组件面板的 Win31 页标签，也就是说，这些对象都是从 Delphi 1.0 的时代就已经存在的。从本小节起，笔者将陆续为大家介绍 TShellComboBox、TShellTreeView、TShellListView 及 TShellChangeNotifier 等对象，这些对象都位于组件面板的 Samples 页标签中。

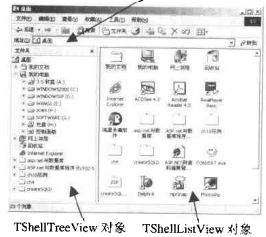


图 12.6　资源管理器

TShellComboBox、TShellTreeView 与 TShellListView 等对象在实务上可以协助您完成“资源管理器”的制作。从图 12.6 中，您可以清楚的了解 TShellComboBox、TShellTreeView 与 TShellListView 等对象在“资源管理器”中分别代表的意义。另外，TShellChangeNotifier 对象则是一个了不起的新对象，它可以监控指定目录下的所有文件或子目录是否被更动或修改，

这对于编写商业程序有很大的帮助。

TShellComboBox 对象继承自 TComboBox 对象，因此有很多属性、操作方法及事件处理都与 TComboBox 对象相同。关于这个部分笔者就不再赘述，以下仅就 TShellComboBox 对象常用的属性加以说明，分述如下：

Root 属性

Root 属性可决定 TShellComboBox 对象所能显示的范围，也就是所要显示的根目录。用户无法选择其他超过 Root 属性的磁盘或目录。Root 属性可以指定某特定目录，以作为 TShellComboBox 对象所要显示的根目录。特定的目录包括：使用系统默认的标准目录(Use Standard Folder)与使用指定的目录(Use Path)等。所谓系统默认的标准目录包括桌面、我的电脑、网上邻居及我的文档等。而指定目录则代表使用文件系统的完整路径名称，例如 C:\temp 及 D:\等。

接着将以表格方式说明 Delphi 6 所提供的标准目录，这里测试的操作系统为 Windows 2000 Server，登录的用户帐号为 administrator，如表 12.5 所示。

表 12.5

系统默认标准目录	实际代表路径或位置
rfDesktop	C:\Documents and Settings \ Administrator\桌面
rfMyComputer	我的电脑
rfNetwork	网上邻居
rfRecycleBin	回收站
rfAppData	C:\Documents and Settings\ Administrator\ Application Data
RfCommonDesktopDirectory	C:\Documents and Settings\All Users\桌面
rfCommonPrograms	C:\Documents and Settings\All Users\ “开始” 菜单\程序
rfCommonStartMenu	C:\Documents and Settings\All Users\ “开始” 菜单
rfCommonStartup	C:\Documents and Settings\All Users\ “开始” 菜单\程序\启动
rfControlPanel	控制面板
rfDesktopDirectory	C:\Documents and Settings\ Administrator\桌面
rfFavorites	C:\Documents and Settings\ Administrator\ Favorites
rfFonts	C:\WINNT\Fonts
rfInternet	Internet Explorer
rfPersonal	C:\Documents and Settings\ Administrator\My Documents
rfPrinters	打印机
rfPrintHood	C:\Documents and Settings\ Administrator\ PrintHood
rfPrograms	C:\Documents and Settings\ Administrator\ “开始” 菜单\程序
rfRecent	C:\Documents and Settings\ Administrator\ Recent
rfSendTo	C:\Documents and Settings\ Administrator\ SendTo
rfStartMenu	C:\Documents and Settings\ Administrator\ “开始” 菜单

续上表

系统默认标准目录	实际代表路径或位置
rfStartup	C:\Documents and Settings\ Administrator\ “开始” 菜单\程序\启动
rfTemplates	C:\Documents and Settings\ Administrator\Templates

- *ShellListView* 属性

ShellListView 属性可连接到一个 TShellListView 对象，并通过 TShellListView 对象显示该指定目录下的所有目录与文件。

- *ShellTreeView* 属性

ShellTreeView 属性可连接到一个 TShellTreeView 对象，并通过 TShellTreeView 对象显示 TShellComboBox 对象指定的根目录(Root 属性)下的所有目录与文件。

- *UseShellImages* 属性

UseShellImages 属性可决定 TShellComboBox 对象是否在每一个数据项前面使用系统默认的小图标。当 UseShellImages 属性值为 True 表示使用；反之，则代表不使用。其默认值为 True，如图 12.7 所示。



图 12.7　UseShellImages 属性

- *Path* 属性

Path 属性可获取指定数据项中的实际目录路径。

- *Items* 与 *ItemIndex* 属性

Items 与 ItemIndex 属性是继承自 TComboBox 对象，可协助您获取指定项目的标题文字。若要获取实际目录路径，则必须使用 Path 属性。有关 Items 与 ItemIndex 属性的详细使用方法，请参考第 8.4 节的详细说明。

范例 12-6

目的：本范例主要说明如何使用 TShellComboBox 对象。

1. 创建新的工程。
2. 在 TForm 对象上加入一个 TShellComboBox 对象与一个 TEdit 对象，其相关属性设

置如表 12.6 所示。

表 12.6

对象	属性名称	属性设置值
TShellComboBox	Name	ShellComboBox1
TEdit	Name	Edit1

3 在 TShellComboBox 对象的 onChange 事件处理过程中加入以下的代码，这段代码主要是获取 TShellComboBox 对象目前所代表的路径位置。

```
procedure TForm1.ShellComboBox1Change(Sender: TObject);
begin
  Edit1.Text := ShellComboBox1.Path;
end;
```

4 保存并运行工程。工程运行后，您可以指定 TShellComboBox 对象中的任何一个目录。此时系统会将其实际的目录位置显示于 Edit1 对象，如图 12.8 所示。

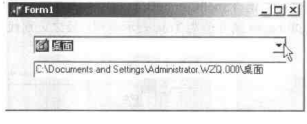


图 12.8 TShellComboBox 对象应用范例

12.6 TShellTreeView 对象

TShellTreeView 对象与 TShellComboBox 对象具有相同的功能，同样都是以树状的方式显示指定目录下的所有子目录或文件。最大的不同在于 TShellTreeView 对象是继承自 TTreeView 对象，而 TShellComboBox 对象则是继承自 TComboBox 对象。因为 TShellTreeView 对象继承自 TTreeView 对象，因此有很多属性、操作方法及事件处理都与 TTreeView 对象相同。关于这个部分笔者就不再赘述，以下仅就 TShellTreeView 对象常用的属性加以说明，分述如下：

Root 属性

Root 属性可决定 TShellTreeView 对象所能显示的范围，也就是所要显示的根目录。用户无法选择其他超过 Root 属性的磁盘或目录。有关 Root 属性的设置方式请参考第 12.5 节的详细说明。

Path 属性

Path 属性可获取指定数据项中的实际目录路径。

Selected 属性

Selected 属性可以获取指定的数据项，其返回值为 TTreeNode 对象。通过 TTreeNode 对象的 Text 属性可获取该数据项的标题文字。有关 TTreeNode 对象请参考第 8.9 节的详细说明。

TopItem 属性

TopItem 属性可以获取目前位于 TShellTreeView 对象最上层的数据项，其返回值为 TTreeNode 对象。通过 TTreeNode 对象的 Text 属性可获取该数据项的标题文字。

ShellComboBox 属性

ShellComboBox 属性可连接到一个 TShellComboBox 对象。

ShellListView 属性

ShellListView 属性可连接到一个 TShellListView 对象，通过 TShellListView 对象可以显示该指定目录下的所有文件。

UseShellImages 属性

UseShellImages 属性可决定 TShellTreeView 对象是否在每一个数据项前面使用系统默认的小图标。当 UseShellImages 属性值为 True 表示使用；反之，则代表不使用。其默认值为 True。

范例 12-7

目的：本范例主要说明如何使用 TShellTreeView 对象。

STEP 1　创建新的工程。

STEP 2　在 TForm 对象上加入一个 TShellTreeView 对象、三个 TLabel 及三个 TEdit 对象，其相关属性设置如表 12.7 所示。

表 12.7

对象	属性名称	属性设置值
TShellTreeView	Name	ShellTreeView1
TLabel	Name	Label1
	Caption	Path
TLabel	Name	Label2
	Caption	Selected
TLabel	Name	Label3
	Caption	TopItem
TEdit	Name	EditPath
TEdit	Name	EditSelected
TEdit	Name	EditTopItem

③ 在 TShellTreeView 对象的 onChange 事件处理过程中加入以下的代码，这段代码主要是获取 TShellTreeView 对象目前所代表的路径位置(Path)、点选的目录(Selected)及最上层的目录(TopItem)等。

```
procedure TForm1.ShellTreeView1Change(Sender: TObject; Node: TTreeNode);
begin
  EditPath.Text := ShellTreeView1.Path;
  EditSelected.Text := ShellTreeView1.Selected.Text;
  EditTopItem.Text := ShellTreeView1.TopItem.Text;
end;
```

④ 保存并运行工程。工程运行后，您可以指定 TShellTreeView 对象中的任何一个目录。此时系统会将其实际的目录位置、点选的目录及最上层的目录显示于 TEdit 对象，如图 12.9 所示。

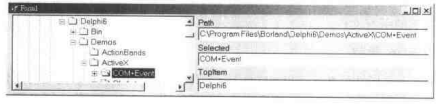


图 12.9 TShellTreeView 对象应用范例

12.7 TShellListView 对象

TShellListView 对象可以显示指定目录下的所有文件或子目录，通常配合 TShellTreeView 对象或 TShellComboBox 对象一起使用。因为 TShellListView 对象继承自 TListView 对象，因此有很多属性、操作方法及事件处理都与 TListView 对象相同。关于这个部分笔者就不再赘述，以下仅就 TShellListView 对象常用的属性加以说明，分述如下。

- *Root 属性*

Root 属性可决定 TShellListView 对象所能显示的范围，也就是所要显示的根目录。用户无法选择其他超过 Root 属性的磁盘或目录。有关 Root 属性的设置方式请参考第 12.5 节的详细说明。

- *ObjectTypes 属性*

ObjectTypes 属性可决定 TShellListView 对象所能显示的文件类型，其包含目录(otFolders)、非目录(otNonFolders)及隐藏文件(otHidden)等。其默认值不包含隐藏文件。

- *ShellComboBox 属性*

ShellComboBox 属性可连接到一个 TShellComboBox 对象。

- *ShellTreeView 属性*

ShellTreeView 属性可连接到一个 TShellTreeView 对象。

ViewStyle 属性

ViewStyle 属性是用来决定 TShellListView 对象显示的方式，共有四种不同的显示方式，如大图标(vsIcon)、小图标(vsSmallIcon)、列表(vsList)及详细资料(vsReport)。

AutoContextMenus 属性

AutoContextMenus 属性可决定 TShellListView 对象中所显示的文件或目录是否支持系统所默认的鼠标右键菜单(Popup Menu)。当 AutoContextMenus 属性值为 True 表示支持；反之，则表示不支持。另外，当您在 TShellListView 对象按下鼠标右键时会触发 onOnContextPopup 事件，如图 12.10 所示。

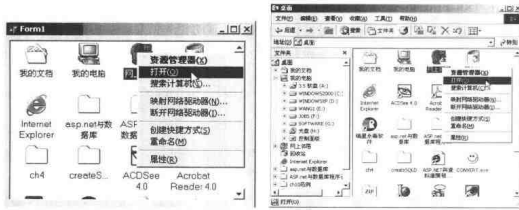


图 12.10　AutoContextMenus 属性

SelectedFolder 属性

SelectedFolder 属性可返回 TShellListView 对象，目前被选取的子目录或文件，其返回值为 TShellFolder 对象。



TShellFolder 对象代表 TShellListView 对象中的文件夹或文件，我们可以通过 TShellFolder 对象的 IsFolder 属性判断目前被选取的项目是否为文件夹；SubFolders 属性可判断指定选项是否包含子目录；PathName 属性可获取指定选项的完整路径；DisplayName 属性可获取指定选项的标题名称；另外，Rename 操作方法则是可以为指定项目更新显示的标题名称，假如其返回值为 True 表示更改名称成功；反之，则代表更改名称失败。TShellFolder 对象的 Rename 操作方法通常配合 TShellListView 对象的 Refresh 操作方法一起使用，以达到实时显示更改名称的结果。

12-8

目的：本范例主要说明如何使用 TShellListView 对象。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TShellTreeView 对象、一个 TShellListView 对象及一个 TMemo 对象，相关属性设置如表 12.8 所示。

表 12.8

对象	属性名称	属性设置值
TShellTreeView	Name	ShellTreeView1
TShellListView	Name	ShellListView1
	ShellTreeView	ShellTreeView1
TMemo	Name	Memo1

步骤 3 在 TShellListView 对象的 onClick 事件处理过程中加入以下的代码，这段代码主要是获取 TShellListView 对象目前所选取数据项的相关信息。

```
procedure TForm1.ShellListView1Click(Sender: TObject);
begin
  if ShellListView1.SelectedFolder <> nil then begin
    Memo1.Clear;
    //标题名称
    Memo1.Lines.Append('Display Name=' +
        ShellListView1.SelectedFolder.DisplayName);
     //是否为文件夹
    Memo1.Lines.Append('Is Folder=' +
        BoolToStr(ShellListView1.SelectedFolder.IsFolder,True));
    //是否包含子目录
    Memo1.Lines.Append('SubFolders=' +
        BoolToStr(ShellListView1.SelectedFolder.

SubFolders,True));
    //获取完整的路径名称
    Memo1.Lines.Append('PathName=' +
        ShellListView1.SelectedFolder.PathName);
  end;
end;
```

注意 笔者在步骤 3 中使用 BoolToStr 函数将 IsFolder 及 SubFolders 等布尔返回值转为字符串类型。BoolToStr 函数的第一个参数代表被转换的布尔值参数；第二个参数决定 BoolToStr 函数的返回值类型。假如第二个参数为 True，那么该函数的返回值为'True'或'False'等字符串；反之，则分别返回'-1'或'0'等字符串

步骤 4 保存并运行工程。运行后，您可以点选 TShellListView 对象中的任何一个数据项。此时系统会在 Memo1 对象上显示其相关信息，如图 12.11 所示。

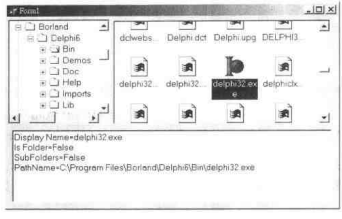


图 12.11　TShellListView 对象应用范例

- *MultiSelect 属性*

MultiSelect 属性可决定 TShellListView 对象是否允许用户一次选取多个子目录或文件。

- *Folders 属性*

Folders 属性包含 TShellListView 对象所有的子目录或文件，您可以通过索引值获取特定的子目录或文件。其返回值为 TShellFolder 对象。

- *Refresh 操作方法*

Refresh 操作方法可重新获取 TShellListView 对象的内容值，通常使用于 TShellListView 对象的内容值被更改时。

范例 12-9

目的：本范例主要说明如何使用 TShellListView 对象的 Folders 属性。

1. 打开范例 12-8。
2. 将原有的 TShellListView 对象 MultiSelect 属性值设为 True。
3. 在 TForm 对象上加入一个 TButton 对象，并为此 TButton 对象的 onClick 事件处理过程加入以下的代码，这段代码主要是获取 TShellListView 对象目前已被选取数据项的路径信息。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  Memo1.Clear;
  For I := 0 to ShellListView1.Items.Count - 1 do begin
     if ShellListView1.Items[i].Selected = True then
       Memo1.Lines.Append(ShellListView1.
```

```
Folders[i].PathName);
  end;
end;
```



在上一个步骤中，笔者使用 TShellListView 对象的 Items 属性，获取所有的数据项个数，以及判断哪些数据项已被选取。最后再利用 Folders 属性获取每一个数据项的完整路径位置。有关 Items 属性请参考第 8.10 节的详细说明。

步骤 4 保存并运行工程。工程运行后，您可以在 TShellListView 对象上选取多个数据项。最后再按下 Button1 按钮获取被选取数据项的完整路径位置，如图 12.12 所示。

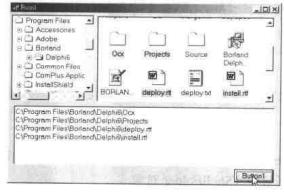


图 12.12 TShellListView 对象应用范例

12.8 TShellChangeNotifier 对象

TShellChangeNotifier 对象可以用来监控某个特定目录，以便得知该特定目录所包含的子目录及文件是否被更动。这些更动包含文件名称被更改、目录名称被更改、目录或文件的属性被更改、文件大小被更改、写入或安全性被更改等。以下就 TShellChangeNotifier 对象常用的属性与事件分述如下：

- *Root 属性*

Root 属性可以指定某特定目录以作为 TShellChangeNotifier 对象所要监控的目录。有关 Root 属性的设置方式请参考第 12.5 节的详细说明。

- *WatchSubTree 属性*

WatchSubTree 属性决定 TShellChangeNotifier 对象所要监控的目录是否包含该目录下的所有子目录。当 WatchSubTree 属性为 True 时表示包含所有的子目录；反之，则不包含所有的子目录。其默认值为 True。

- *NotifyFilters 属性*

NotifyFilters 属性决定 TShellChangeNotifier 对象所要监控更新的种类，包括：文件名称

被更改(nfFileNameChange)、目录名称被更改(nfDirNameChange)、目录或文件的属性被更改(nfAttributeChange)、文件大小被更改(nfSizeChange)及写入或安全性被更改(nfWriteChange 与 nfSecurityChange)等。其默认值仅包含 nfFileNameChange 与 nfDirNameChange 等。

onChange 事件

当 TShellChangeNotifier 对象所指定的目录中发生符合 NotifyFilters 属性所设置的更新种类时，系统会触发一个 TShellChangeNotifier 对象的 onChange 事件。

范例 12-10

目的：本范例主要说明 TShellChangeNotifier 对象的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TShellChangeNotifier 对象及一个 TMemo 对象，其相关属性设置如表 12.9 所示。

表 12.9

对象	属性名称	属性设置值
TShellChangeNotifier	Name	ShellChangeNotifier1
	WatchSubTree	False
	NotifyFilters	[nfFileNameChange]
TMemo	Name	Memo1

步骤 3 在 TShellChangeNotifier 对象的 onChange 事件处理过程中加入以下的代码，这段代码主要是仿真一个数据交换的处理过程。我们假设客户将订单数据以 FTP 或 EDI 的方式将其上传 FTP 服务器，也就是 TShellChangeNotifier 对象所监控的目录中。这个程序则实时的将这些数据加以处理，通常是将其保存在数据库中。最后，再将其移动到备份的目录，以表示工作的完成。

```
procedure TForm1.ShellChangeNotifier1Change;
var
   Srec : TSearchRec;
   Path, BackupPath ,SourceFile,BackupFile : String;
begin
   Memo1.Lines.Append('onChange-Start');
   //获取工作目录与备份目录
   Path := ShellChangeNotifier1.Root;
   BackupPath := Path + 'Backup\';
   //判断备份目录是否存在
   if NOT DirectoryExists(BackupPath) then
     ForceDirectories(BackupPath);
   //查找指定目录下符合条件的文件
```

```
    if FindFirst(Path + '*.txt', faAnyFile, Srec) = 0 then begin
      repeat
        //获取目前文件的完整路径
        SourceFile := Path + Srec.Name;
        //设置备份文件的完整路径
        BackupFile := BackupPath +ChangeFileExt(Srec.Name,'.BAK');
        //进行您指定的工作
        //(省略)
        //将已处理的文件移动到备份目录
        MoveFile(PChar(SourceFile),PChar(BackupFile));
      //继续查找下一个符合条件的文件
      until FindNext(Srec) <> 0;
      //结束查找
      FindClose(Srec);
    end;
    Memo1.Lines.Append('onChange-End');
  end;
```

Step 4 保存工程。

Step 5 按【F9】功能键运行工程。工程运行后，您可以通过资源管理器复制文件到 TShellChangeNotifier 对象所监控的目录，此时系统会实时地触发 TShellChangeNotifier 对象的 onChange 事件。并且将符合条件的文件加以移动到备份目录，如图 12.13 所示。

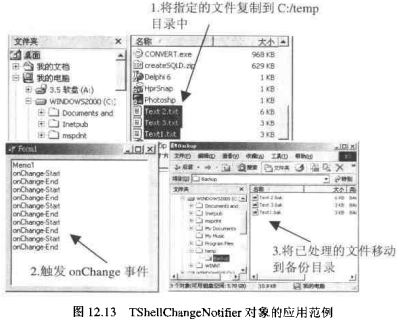


图 12.13 TShellChangeNotifier 对象的应用范例

第13章

Delphi 6 图像与绘图管理对象

本章目标

本章将为您介绍 Delphi 6 所提供的图像与绘图管理对象，本章所要介绍的图像与绘图对象共有 11 个，请参考各小节的详细说明。

本章主题

13.1 Tcanvas 对象

13.2 TShape 对象

13.3 TImage 对象

13.4 TImageList 对象

13.5 TPaintBox 对象

13.6 TBevel 对象

13.7 TScreen 对象

13.8 TColorGrid 对象

13.9 TColorBox 对象

13.10 TAnimate 对象

13.11 TMediaPlayer 对象

13.1 TCanvas 对象

TCanvas 对象是 Delphi 所提供的一个绘图对象，您可以想象它就像是一张画布，它可以附属在很多的对象上。例如 TForm 对象的 Canvas 属性就是对应到一个 TCanvas 对象，因此您可以想象 TForm 对象上有一张透明的画布。一张画布通常与下列几项工具有关：

- 画笔：TCanvas 对象的 Pen 属性对应于一个 TPen 对象，它就是一个画笔对象，其用来控制画布任何绘图操作的线条效果，例如线条的粗细及样式。请参考第 13.1.1 节的详细说明。
- 笔刷：TCanvas 对象的 Brush 属性对应于一个 TBrush 对象，它就是一个笔刷对象，其用来控制画布上任何绘图操作的图形、背景颜色、样式，请参考第 13.1.2 节的详细说明。
- 字体：TCanvas 对象的 Font 属性对应于一个 TFont 对象，它用来控制字体种类、颜色、大小和样式。请参考第 13.1.3 节的详细说明。
- 画笔位置：TCanvas 对象的 PenPos 属性对应于一个 TPoint 类型，它用来决定画笔的坐标位置。
- 像素：TCanvas 对象的 Pixels 属性可返回指定区域的像素颜色，也就是说 Pixels 属性可返回指定“点”的颜色。

上面所介绍的是与画布有关的几个工具，事实上，这也是 TCanvas 对象常用的属性。以下将以实例说明如何通过 TCanvas 对象的 Pixels 属性制作一个类似于屏幕保护程序的范例，此范例会在一个黑色的画布上以随机的方式画上白色的点。

范例 13-1

目的：本范例将说明如何通过 TCanvas 对象的 Pixels 属性来改变画面每个点(Pixel)的颜色。

步骤 1 创建新的工程。

步骤 2 将默认的 TForm 对象的 Color 属性改为 clBlack。

步骤 3 在 TForm 对象上加入一个 TTimer 对象(在 System 组件面板上)。将其 Interval 属性设置 1(也就是说千分之一秒运行一次下列的程序)，并在 TTimer 对象的 onTimer 事件处理过程中加入以下的代码：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  //不规则改变一个点的颜色
  Canvas.Pixels[Random(Width),Random(Height)] := clWhite;
end;
```

步骤 4 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //随机数生成器
```

```
    Randomize;
    //启动 Timer 对象
    Timer1.Enabled := True;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】键运行工程。工程运行后，您会发现 TForm 对象的颜色开始呈不规则的变化，直到整个画面由黑色变白色为止。

接下来我们来谈谈 Delphi 为 TCanvas 对象画布提供哪些画图的方法，也就是 TCanvas 对象所提供的操作方法，分述如下：

Arc 操作方法

Arc 操作方法可以让您在 TCanvas 对象上画一个弧形，此参数需要三个坐标点，其声明语法如下：

```
procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
```

Pie 操作方法

Pie 操作方法可以让您在 TCanvas 对象上画一个 Pie 的图形，此参数需要四个坐标点，其声明语法如下：

```
procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Longint);
```

Ellipse 操作方法

Ellipse 操作方法可以让您在 TCanvas 对象上画一个椭圆形，此参数需要两个坐标点，其声明语法如下：

```
procedure Ellipse(X1, Y1, X2, Y2: Integer); overload;
procedure Ellipse(const Rect: TRect); overload;
```

Rectangle 操作方法

Rectangle 操作方法可以让您在 TCanvas 对象上画一个矩形，此参数需要两个坐标点，其声明语法如下：

```
procedure Rectangle(X1, Y1, X2, Y2: Integer); overload;
procedure Rectangle(const Rect: TRect); overload;
```

RoundRect 操作方法

RoundRect 操作方法可以让您在 TCanvas 对象上画一个圆角矩形，此参数需要二个坐标点，其声明语法如下：

```
procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
```

Polygon 操作方法

Polygon操作方法可以让您在TCanvas对象上画一个多角(边)形，此参数需要多个点坐标，结束的坐标不需要和开始的坐标相同，其声明语法如下：

```
procedure Polygon(Points: array of TPoint);
```

Polyline 操作方法

Polyline操作方法可以让您在TCanvas对象上画一个多角(边)线，此参数需要多个点坐标，其声明语法如下：

```
procedure Polyline(Points: array of TPoint);
```

FillRect 操作方法

FillRect操作方法可以让您在TCanvas对象填充指定的范围，FillRect操作方法的声明语法如下：

```
procedure FillRect(const Rect: TRect);
```

LineTo 操作方法

LineTo操作方法可以让您从TCanvas对象目前的坐标位置画一直线至指定的坐标位置，LineTo操作方法通常会配合MoveTo操作方法一起使用，其声明语法如下：

```
procedure LineTo(X, Y: Integer);
```

MoveTo 操作方法

MoveTo操作方法可以让您移动TCanvas对象上的坐标位置，其声明语法如下：

```
procedure MoveTo(X, Y: Integer);
```

Chord 操作方法

Chord操作方法可以让您在TCanvas对象上画一个弦形图案，此参数需要四个点坐标，其声明语法如下：

```
procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
```

Draw 操作方法

Draw操作方法可以让您在TCanvas对象指定坐标中显示图形文件，其声明语法如下：

```
procedure Draw(X, Y: Integer; Graphic: TGraphic);
```

TextExtent 操作方法

TextExtent操作方法以点(pixels)的方式返回目前字符串的高度与宽度，其声明语法如下：

```
type
  TSize = record
    cx: Longint;
    cy: Longint;
  end;

function TextExtent(const Text: string): TSize;
```

- *TextHeight* 操作方法

TextHeight 操作方法以点(pixels)的方式返回目前字符串的高度，其声明语法如下：

```
function TextHeight(const Text: string): Integer;
```

- *TextWidth* 操作方法

TextWidth 操作方法以点(pixels)的方式返回目前字符串的宽度，其声明语法如下：

```
function TextWidth(const Text: string): Integer;
```

- *TextOut* 操作方法

TextOut 操作方法可以在指定的坐标显示字符串，其声明语法如下：

```
procedure TextOut(X, Y: Integer; const Text: string);
```



13-2

目的：本范例将说明如何使用 TCanvas 对象所提供的操作方法绘制固定形状的图形。

STEP 1　创建新的工程。

STEP 2　在 TForm 对象上加入 1 个 TButton 对象，其相关属性设置如表 13.1 所示。

表 13.1

对象	属性名称	属性设置值
TButton	Name	btArc
	Caption	Arc
TButton	Name	btPloygon
	Caption	Ploygon
TButton	Name	btPie
	Caption	Pie
TButton	Name	btEllipse
	Caption	Ellipse
TButton	Name	btRectangle
	Caption	Rectangle

续上表

对象	属性名称	属性设置值
TButton	Name	btRoundRect
	Caption	RoundRect
TButton	Name	btPolyline
	Caption	Polyline
TButton	Name	btChord
	Caption	Chord
TButton	Name	btDraw
	Caption	Draw
TButton	Name	btTextOut
	Caption	TextOut

步骤 3 在 btArc 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btArcClick(Sender: TObject);
begin
  //画出一个弧形
  Canvas.pen.Color := clBlue;
  Canvas.pen.width := 3;
  Canvas.Arc(300, 300, 470, 250, 400, 220, 300, 300);
end;
```

步骤 4 在 btPolygon 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btPolygonClick(Sender: TObject);
begin
  //画出一个多角(边)形
  Canvas.Brush.Color := clTeal;
  Canvas.Brush.Style := bsDiagCross;
  Canvas.Polygon([Point(200, 70), Point(150, 80),
                  Point(250, 20), Point(440, 120)]);
end;
```

步骤 5 在 btPie 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btPieClick(Sender: TObject);
begin
   //画出一个 Pie 的图形
   Canvas.Brush.Color := clYellow;
   Canvas.Brush.Style := bsCross;
   Canvas.Pie(100, 100, 200, 200, 130, 130, 150, 150);
end;
```

步骤 6 在 btEllipse 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btEllipseClick(Sender: TObject);
```

```
begin
    //画出椭圆形
    Canvas.Brush.Color := clGreen;
    Canvas.Brush.Style := bsCross;
    Canvas.Ellipse(150, 320, 80, 210);
end;
```

STEP 7 在 btRectangle 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btRectangleClick(Sender: TObject);
begin
    //画出矩形
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsBDiagonal;
    canvas.Rectangle(520,50,600,90);
end;
```

STEP 8 在 btRoundRect 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btRoundRectClick(Sender: TObject);
begin
    //画出圆角方形
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsBDiagonal;
    canvas.RoundRect(420,50,500,90,20,20);
end;
```

STEP 9 在 btPolyline 对象的 onClick 事件处理过程中加入下列代码：

```
procedure TForm1.btPolylineClick(Sender: TObject);
begin
  //画出多角(边)线
  Canvas.pen.Color := clLime;
  Canvas.Polyline([Point(250,220),Point(230,270),Point(280,240),
  Point(220, 240), Point(270, 270), Point(250, 220)]);
end;
```

STEP 10 在 btChord 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btChordClick(Sender: TObject);
begin
    //画出弦形图案
    canvas.chord(0,0,100,100,50,50,10,10);
end;
```

STEP 11 在 btDraw 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btDrawClick(Sender: TObject);
var
  Bitmap : TBitMap;
```

```
begin
    //显示图形文件
    //方法一
    Bitmap := TBitmap.Create;
    BitMap.LoadFromFile('c:\Program Files\Common Files\Borland Shared\
Images\Splash\256color\factory.bmp');
    Canvas.Draw(400,150,BitMap);
    Bitmap.Free;
    //方法二
    {
    Canvas.Brush.Bitmap := TBitmap.Create;
    Canvas.Brush.Bitmap.LoadFromFile('c:\Program Files\Common Files\Borland
Shared\Images\Splash\256color\factory.bmp');
    Canvas.Draw(300,150,Canvas.Brush.Bitmap);
    Canvas.Brush.Bitmap.Free;
    Canvas.Brush.Bitmap := nil;
    }
end;
```

STEP 12 在 btTextOut 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btTextOutClick(Sender: TObject);
begin
  //显示文字
  canvas.Font.size := 30;
  canvas.textout(300,1,'Delphi 6 实务经典');
end;
```

STEP 13 保存工程，并按下【F9】功能键运行工程。工程运行后，您可以通过这些 TButton 对象来画出各种不同的图案，如图 13.1 所示。

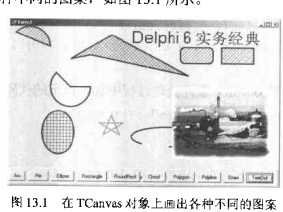


图 13.1 在 TCanvas 对象上画出各种不同的图案

13.1.1 TPen 画笔对象

前面笔者曾经说过画笔可控制画布任何绘图操作的线条效果，例如线条对象的粗线及样

式，本小节将为您说明画笔对象的相关设置。

- 画笔颜色：TPen 对象的 Color 属性对应于一个 TColor 对象，它可以决定画笔所画出的颜色，有关 TColor 对象请参考第 3 章的详细介绍。
- 画笔宽度：TPen 对象的 Width 属性对应于一个整数变量，它可以决定画笔所画出的宽度。
- 画笔样式：TPen 对象的 Style 属性对应于一个 TPenStyle 对象，它可以决定画笔所画出的样式。TpenStyle 对象包含 psSolid、psDash、psDot、psDashDot、psDashDotDot、psClear 及 psInsideFrame 等不同的设置值。表 13.2 将加以说明这些设置值所代表的意义。

表 13.2

设置值	设置值图标说明
psClear	(无)
psSolid	————
psDash	– – – – –
psDot	
psDashDot	– – . – – .
psDashDotDot	– – . . – – . .
psInsideFrame	————

- 画笔模式：TPen 对象的 Mode 属性对应于一个 TPenMode 对象，它可以决定画笔所画出的模式。TPenMode 对象包含 pmBlack、pmWhite、pmNop、pmNot、pmCopy、pmNotCopy、pmMergePenNot、pmMaskPenNot、pmMergeNotPen、pmMaskNotPen、pmMerge、pmNotMerge、pmMask、pmNotMask、pmXor 及 pmNotXor 等不同的设置值。表 13.3 将加以说明这些设置值所代表的意义。

表 13.3

设置值	说明
pmBlack	显示颜色为黑色，无论 Color 属性值为何
PmWhite	显示颜色为白色，无论 Color 属性值为何
PmNop	无变化的(Unchanged)
PmNot	显示颜色为背景颜色的反向(inverse)
PmCopy	画笔(Pen)颜色由 Color 属性来决定，这也是默认值
PmNotCopy	画笔(Pen)颜色为 Color 属性的反向颜色
PmMergePenNot	结合 Color 属性与背景的反向颜色
PmMaskPenNot	结合 Color 属性与背景的反向颜色的共同部分
PmMergeNotPen	结合背景颜色与 Color 属性的反向颜色
PmMaskNotPen	结合背景颜色与 Color 属性的反向颜色的共同部分
PmMerge	结合 Color 属性与背景颜色

续上表

设置值	说明
pmNotMerge	pmMerge 的反向颜色，也就是结合 Color 属性与背景颜色的反向颜色
pmMask	结合背景颜色与 Color 属性颜色的共同部分
pmNotMask	pmMask 的反向颜色，也就是结合背景颜色与 Color 属性颜色的共同部分的反向颜色
pmXor	对 Color 属性与背景颜色做 XOR
pmNotXor	pmXor 的反向颜色，也就是对 Color 属性与背景颜色做 XOR 再反向的颜色

13.1.2 TBrush 笔刷对象

前面笔者曾经说过笔刷可控制画布上任何绘图操作的图形或背景颜色与样式，本小节将为您说明笔刷对象的相关设置。

- 笔刷颜色：TBrush 对象的 Color 属性对应于一个 TColor 对象，它可以决定笔刷的颜色，有关 TColor 对象请参考第 3 章的介绍。
- 笔刷样式：TBrush 对象的 Style 属性对应于一个 TBrushStyle 对象，它可以决定笔刷的样式。TBrushStyle 对象包含 bsSolid、bsClear、bsHorizontal、bsVertical、bsFDiagonal、bsBDiagonal、bsCross 及 bsDiagCross 等不同的设置值。表 13.4 将加以说明这些设置值所代表的意义。

表 13.4

设置值	设置值图标说明
bsClear	(无)
bsSolid	
bsHorizontal	
bsVertical	
bsFDiagonal	
bsBDiagonal	
bsCross	
bsDiagCross	

- 图形样式：TBrush 对象的 Bitmap 属性对应于一个 TBitmap 对象，它可以使用位图作为 TBrush 对象的笔刷样式。



13-3

目的：本范例将说明如何使用 TBrush 对象的 Bitmap 属性在 TForm 对象上显示一个图形文件。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TButton 对象，并在 onClick 事件处理过程中加入以下的

代码。这段代码主要在设置 TBrush 对象的 Bitmap 属性值并显示其图形文件。

```
procedure TForm1.Button1Click(Serder: TObject);
var
  Bitmap : TBitmap;
begin
  try
    //创建一个 TBitmap 对象
    Bitmap := TBitmap.Create;
    //加载指定的图形文件
    Bitmap.LoadFromFile('FACTORY.BMP');
    //设置笔刷的样式
    Form1.Canvas.Brush.Bitmap := Bitmap;
    //画出一个与图形文件同样大小的方形
    Form1.Canvas.FillRect(Rect(0, 0 ,
                                  Bitmap.Width, Bitmap.Height));
  finally
    //取消笔刷的样式
    Form1.Canvas.Brush.Bitmap := nil;
    //释放 TBitmap 对象
    Bitmap.Free;
  end;
end;
```

③ 保存并运行工程。工程运行后，当您按下 Button1 按钮时，系统会加载指定的图形文件并加以显示于 TForm 对象上，如图 13.2 所示。



图 13.2　TBrush 对象的 Bitmap 属性运行范例

13.1.3　TFont 字体对象

TFont 是一个字体对象，很多的对象都包含这个属性，例如：TLabel、TEdit、TMemo、TComboBox、TPanel 及 TCanvas 等。以下就 TFont 对象常用的属性加以说明。

- Charset 属性是用来设置 Font 的字符集(character set)，其值介于 0~255 之间，以中文而言其值为 136、常数为“CHINESEGB2312_CHARSET”。至于其他的字符集请参考联机帮助文件。

- Color 属性是用来设置 Font 的文字颜色，其继承自 TColor 对象。有关 TColor 对象请参考第 3 章的详细介绍。
- Height 属性是用来设置 Font 的文字高度。
- Size 属性是用来设置 Font 的文字大小。
- Style 属性是用来设置 Font 的字体样式，其继承自 TFontStyle 数据类型。TFontStyle 数据类型包含 fsBold、fsItalic、fsUnderline 及 fsStrikeOut 等可能的值，其分别代表粗体、斜体、下划线及删除线。
- Name 属性是用来设置 Font 的字体名称，您可以通过 TScreen 对象的 Fonts 属性获取系统的字体名称。

13.2 TShape 对象

TShape 对象位于 Additional 的组件面板上，它是一个无法获取焦点(Focus)的对象。在范例 4-5 中我们曾经使用 TShape 对象作为我们所画的边框，也因为如此，我们才能移动这个边框。由于 TShape 对象与其他对象一样有很多相同的属性，这些相同的属性您可以参考本书第 3 章的说明，本小节仅就 TShape 对象特有的属性加以说明。

Brush 属性

Brush 属性是继承自 TBrush 对象，TBrush 对象的 Color 属性与 Style 属性分别代表 TShape 对象的底纹颜色与底纹格式。关于 TBrush 对象请参考第 13.1.2 节的详细说明。

Pen 属性

Pen 属性继承自 TPen 对象，TPen 对象的 Color 属性、Mode 属性、Style 属性及 Width 属性分别代表 TShape 对象框线的颜色、画线方法、框线类型及框线粗细等。有关 TPen 对象请参考第 13.1.1 节的详细说明。

Shape 属性

Shape 属性代表 TShape 对象的外观类型，继承自 TShapeType 数据类型。其包含 stRectangle、stSquare、stRoundRect、stRoundSquare、stEllipse 及 stCircle 等不同的设置值。表 13.5 将加以说明这些设置值所代表的意义。

表 13.5

设置值	说明	设置值图标说明
stRectangle	矩形或长方形	
stSquare	正方形	
stRoundRect	圆角矩形	
stRoundSquare	圆角方形	
stEllipse	椭圆形	
stCircle	圆形	

范例 13-4

目的：本范例示范 TShape 对象常用的各种属性。本范例允许用户设置 TShape 对象的各种属性，再根据这些属性值动态产生 TShape 对象。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象上加入一个 TColorDialog 对象及一个 TPanel 对象。并且在 TPanel 对象中加入七个 TLabel 对象、六个 TcomboBox 对象与一个 TSpanEdit 对象，对象摆放请参考图 13.3，其相关属性设置如表 13.6 所示。

表 13.6

对象	属性名称	属性设置值
TColorDialog	Name	ColorDialog1
TPanel	Name	Panel1
TLabel	Caption	Shape Type
TLabel	Caption	Brush Style
TLabel	Caption	Brush Color
TLabel	Caption	Pen Style
TLabel	Caption	Pen Color
TLabel	Caption	Pen Mode
TLabel	Caption	Pen Width
TComboBox	Name	cbShapeType
	Style	csDropDownList
	Items	(请输入所有的 Shape Type)
TComboBox	Name	cbBrushStyle
	Style	csDropDownList
	Items	(请输入所有的 Brush Type)
TComboBox	Name	cbBrushColor
	Style	csDropDownList
	Items	(请输入所有的 Brush Color)
TComboBox	Name	cbPenStyle
	Style	csDropDownList
	Items	(请输入所有的 Pen Style)
TComboBox	Name	cbPenColor
	Style	csDropDownList
	Items	(请输入所有的 Pen Color)
TComboBox	Name	cbPenMode
	Style	csDropDownList
	Items	(请输入所有的 Pen Mode)

续上表

对象	属性名称	属性设置值
TSpanEdit	Name	sePenWidth
	MaxValue	20
	MinValue	1

步骤 3 本范例需要动态产生 TShape 对象，因此我们必须先在 Uses 句子中加入 ExtCtrls 单元文件，因为 TShape 对象被定义在此单元中。

步骤 4 在 TForm1 对象的私有变量区(Private)声明 v_BrushColor 等六个变量。因为我们不能直接将 TComboBox 对象的 Text 属性值指定给 TShape 对象相关属性值，因此我们必须通过特定的数据类型来保存用户所设置的各项属性值。

```
private
    { Private declarations }
    //自订变数
    v_BrushColor : String;
    v_PenColor   : String;
    v_ShapeType  : TShapeType;
    v_BrushStyle : TBrushSty_e;
    v_PenStyle   : TPenStyle;
    v_PenMode    : TPenMode;
```

注意 我们不能直接将 TComboBox 对象的 Text 属性值指定给 TShape 对象相关属性值，因此我们必须通过其所对应的数据类型来保存用户所选择的属性值，例如 v_ShapeType、v_BrushStyle、v_PenStyle 与 v_PenMode 等都是为了这种转换目的而设立的。由于 Delphi 6 提供了 StringToColor 与 ColorToString 两个函数让我们在字符串(String)与颜色(TColor)等不同数据类型中方便的进行转换。这两个函数的声明如下：

```
//将 TColor 数据类型转为 String 类型
function ColorToString(Color: TColor): string;

//将 String 类型转为 TColor 数据类型
function StringToColor(const S: string): TColor;
```

步骤 5 在 cbShapeType 对象的 onChange 事件处理过程中加入以下的过程代码：

```
procedure TForm1.cbShapeTypeChange(Sender: TObject);
begin
  //设置 Shape 对象的 Shape Type
  case cbShapeType.ItemIndex of
      0 : v_ShapeType := stRectangle;
      1 : v_ShapeType := stSquare;
      2 : v_ShapeType := stRoundRect;
      3 : v_ShapeType := stRoundSquare;
```

```
        4 : v_ShapeType := stEllipse;
        5 : v_ShapeType := stCircle;
    end;
end;
```

6 在 cbBrushStyle 对象的 onChange 事件处理过程中加入以下过程代码：

```
Procedure TForm1.cbBrushStyleChange(Sender: TObject);
begin
  //设置 Shape 对象的 Brush Style
  case cbBrushStyle.itemindex of
      0 : v_BrushStyle :  bsSolid;
      1 : v_BrushStyle := bsClear;
      2 : v_BrushStyle := bsHorizontal;
      3 : v_BrushStyle := bsVertical;
      4 : v_BrushStyle := bsFDiagonal;
      5 : v_BrushStyle := bsBDiagonal;
      6 : v_BrushStyle := bsCross;
      7 : v_BrushStyle := bsDiagCross;
  end;
end;
```

7 在 cbBrushColor 对象的 onChange 事件处理过程中加入以下过程代码：

```
procedure TForm1.cbBrushColorChange(Sender: TObject);
begin
   if (Sender as TComboBox).Text = 'OtherColor' then
     if ColorDialog1.Execute then
       v_BrushColor := ColorToString(ColorDialog1.Color);
end;
```

8 在 cbPenStyle 对象的 onChange 事件处理过程中加入以下的过程代码：

```
procedure TForm1.cbPenStyleChange(Sender: TObject);
begin
  //设置 Shape 对象的 Pen Style
  case cbPenStyle.itemindex of
      0 : v_PenStyle := psSolid;
      1 : v_PenStyle := psDash;
      2 : v_PenStyle := psDot;
      3 : v_PenStyle := psDashDot;
      4 : v_PenStyle := psDashDotDot;
      5 : v_PenStyle := psClear;
      6 : v_PenStyle := psInsideFrame;
  end;
end;
```

9 在 cbPenColor 对象的 onChange 事件处理过程中加入以下的过程代码：

```
procedure TForm1.cbPenColorChange(Sender: TObject);
begin
   if (Sender as TComboBox).Text = 'OtherColor' then
     if ColorDialog1.Execute then
       v_PenColor := ColorToString(ColorDialog1.Color);
end;
```

(11) 在 cbPenMode 对象的 onChange 事件处理过程中加入以下的过程代码：

```
procedure TForm1.cbPenModeChange(Sender: TObject);
begin
  //设置 Shape 对象的 Pen Mode
  case cbPenMode.itemindex of
      0 : v_PenMode := pmBlack;
      1 : v_PenMode := pmWhite;
      2 : v_PenMode := pmNop;
      3 : v_PenMode := pmNot;
      4 : v_PenMode := pmCopy;
      5 : v_PenMode := pmNotCopy;
      6 : v_PenMode := pmMergePenNot;
      7 : v_PenMode := pmMaskPenNot;
      8 : v_PenMode := pmMergeNotPen;
      9 : v_PenMode := pmMaskNotPen;
     10 : v_PenMode := pmMerge;
     11 : v_PenMode := pmNotMerge;
     12 : v_PenMode := pmMask;
     13 : v_PenMode := pmNotMask;
     14 : v_PenMode := prXor;
     15 : v_PenMode := pmNotXor;
  end;
end;
```

(12) 在 Form1 对象的 onCreate 事件处理过程中加入以下的过程代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //定义各种属性的初值
   cbPenStyle.ItemIndex := 0;
   cbBrushColor.ItemIndex := 11;
   cbPenColor.ItemIndex := 5;
   cbBrushStyle.ItemIndex := 0;
   cbShapeType.ItemIndex := 4;
   cbPenMode.ItemIndex := 4;

   //调用各种属性对象的 onChange 事件
   cbBrushColorChange(nil);
   cbPenColorChange(nil);
```

```
    cbShapeTypeChange(nil);
    cbBrushStyleChange(nil);
    cbPenStyleChange(nil);
    cbPenModeChange(nil);
end;
```

(13) 在 Form1 对象的 onMouseDown 事件处理过程中加入以下的代码：

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  //创建一个新的 Shape 对象
  with TShape.Create(self) do
      begin
        Parent := self;
        width  := 100;
        Height := 80;
        Top    := Y;
        Left   := X;
        //Shape Type
        Shape  := v_ShapeType;
        //Brush Style
        Brush.Style := v_BrushStyle;
        //Pen Width
        Pen.Width   := sePenWidth.Value;
        //Pen Style
        Pen.Style   := v_PenStyle;
        //Pen Mode
        Pen.Mode    := v_PenMode;

        //设置 Brush Color 属性值
        if cbBrushColor.Text = 'OtherColor' then
          Brush.color  := StringToColor(v_BrushColor)
        else
          Brush.color  := StringToColor(cbBrushColor.Text);
        //设置 Pen Color 属性值
        if cbPenColor.Text = 'OtherColor' then
          Pen.color  := StringToColor(v_PenColor)
        else
          Pen.color  := StringToColor(cbPenColor.Text);
      end;
end;
```

(14) 保存工程，并按【F9】功能键运行工程。此时您可以设置即将产生的 TShape 对象的任何属性。设置完成后，您只要使用鼠标在 Form1 对象上点一下，那么 Form1

对象的 onMouseDown 事件即被触发，因此一个新的 TShape 对象也就被产生了，如图 13.3 所示。

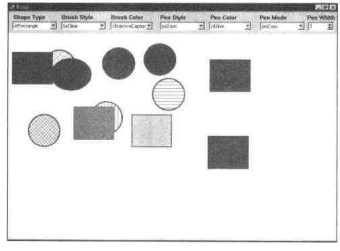


图 13.3 TShape 对象综合范例

13.3 TImage 对象

TImage 对象位于对象面板的 Additional 页标签中，它也是一个容器组件，可用来显示图形文件。以下就 TImage 对象常用的属性及操作方法加以说明。

- *Canvas 属性*

TImage 对象的 Canvas 属性是继承自 TCanvas 对象，有关 TCanvas 对象请参考第 13.1 节的详细说明。

- *Center 属性*

Center 属性是用来决定图形文件是否居中显示。

- *IncrementalDisplay 属性*

IncrementalDisplay 属性用来决定图形文件是否使用渐连式的显示方式。

- *Proportional 属性*

Proportional 属性用来决定图形文件是否以等比例的方式显示。

- *Stretch 属性*

Stretch 属性用来决定图形文件是否按照 TImage 对象的长宽自行调整大小。

- *Transparent 属性*

Transparent 属性用来决定图形文件是否以透明的方式显示。

Picture 属性

Pictrue 属性继承自 TPciture 对象，它允许 TImage 对象显示位图(bitmap)、图标(icon)、中继文件(metafile)或是用户自行定义的图形文件。以下将示范如何在设计时期为 TImage 对象加入一个图形文件。

范例 13-5

目的：本范例将示范如何在设计时期为 TImage 对象加入一个图形文件。

1. 创建新的工程。
2. 在 TForm 对象上加入一个 TImage 对象并点选 Picture 属性，以打开一个 Picture Editor 对话框。
3. 在 Picture Editor 对话框中单击 Load 按钮并选择适当的图形文件。
4. 此时图形文件即被加载到 TImage 对象中，如图 13.4 所示。

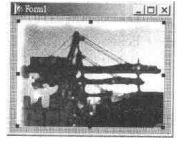


图 13.4　设计时期为 TImage 对象加载图形文件

范例 13-6

目的：本范例将示范如何在运行时期为 TImage 对象加入一个图形文件。

1. 打开范例 13-5。
2. 清除 TImage 对象 Picture 属性已经加载的图形文件，并在 TForm 对象的 onCreate 事件处理过程加入以下的代码。笔者这里示范三个方法，第一个方法通过 TImage 对象 Picture 属性的 LoadFromFile 操作方法加载一个已经存在的图形文件。第二个方法通过 TBitmap 对象创建一个新的图形文件。第三个方法则是通过 TBitmap 对象的 LoadFromFile 操作方法加载一个已经存在的图形文件。

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Bitmap : TBitmap;
begin
  //方法一(加载已经存在的图形文件)
  //Image1.Picture.LoadFromFile('SHIPPING.BMP');
```

```
    //方法二(创建一个新的图形文件)
    Bitmap := TBitmap.Create;
    //Bitmap.Width := Image1.Width;
    //Bitmap.Height := Image1.Height;
    //方法三(加载已经存在的图形文件)
    Bitmap.LoadFromFile('SHIPPING.BMP');
    Image1.Picture.Graphic := Bitmap;
    Bitmap.Free;
end;
```

步骤 3 保存并按【F9】键运行工程。工程运行后，如果您是使用第一个或第三个方法，那么系统会自动为您加载指定的图形文件。如果是第二个方法，系统则为您加载一个背景色为白色的图形文件。

13.4 TImageList 对象

TImageList 对象位于对象面板的 System 页标签中，它也是一个容器组件，可将应用程序中常用的图片加以集中以便达到资源共享与集中管理的目的。您可以在 TImageList 对象上双击、或按下鼠标右键选择“ImageList Editor...”功能选项以打开“图片编辑器”对话框。在图片编辑器中您可以加载多个图片以利其他对象使用，例如 TMainMenu、TPopupMenu 或 ToolBar 等，如图 13.5 所示。

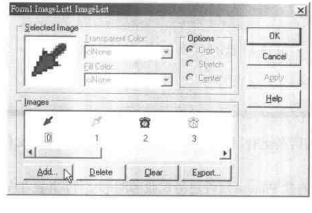


图 13.5 设计时期为 TImageList 对象加载图形文件

以下就 TImageList 对象常用的属性加以说明。

- *AllocBy 属性*

AllocBy 属性用来决定 TImageList 对象每次分配内存的大小，其默认值为 4。

- *BkColor 属性*

BkColor 属性用来决定 TImageList 显示图片时的背景颜色。

- *BlendColor 属性*

 BlendColor 属性是用来决定 TImageList 显示图片时的前景颜色。

- *Count 属性*

 Count 属性是一个运行时期属性，它可以获取 TImageList 对象中的图案个数。

- *Height 属性*

 Height 属性是用来限定 TImageList 对象所包含的图形文件高度。

- *Width 属性*

 Width 属性是用来限定 TImageList 对象所包含的图形文件宽度。

- *DrawingStyle 属性*

 DrawingStyle 属性是用来决定图片显示的形式，其包含 dsNormal、dsTransparent、dsFocus 及 dsSelected 等。dsNormal 表示图片显示不做任何的变化处理；dsTransparent 表示图片显示不因 bkColor 属性值而改变；dsFocus 表示当图标显示时会混合 25%的背景颜色，但必须配合 Masked 属性值为 True 才有作用；dsSelected 表示当图标显示时会混合 50%的背景颜色，但必须配合 Masked 属性值为 True 才有作用。

- *Masked 属性*

 Masked 属性决定此 TImageList 对象中的图片显示时是否具有 Mask 的能力。Masked 属性会直接影响 DrawingStyle 属性的显示结果。

- *ImageType 属性*

 ImageType 属性用来决定图片的类型，其包括 itImage 及 itMask 两种类型。

13.5　TPainBox 对象

TPainBox 对象主要的功能也是用于绘图，它同样可以通过 Canvas 属性来显示图片。使用 TPainBox 对象的 Canvas 属性与 TForm 对象的 Canvas 属性最大的不同在于坐标，前者是按照 TPainBox 对象的坐标，而后者则是使用 TPainBox 对象的坐标。另一个重点是“重画”的问题，前面几个范例中，您会发现由 TCanvas 对象所绘制的图形或图片都会因为窗口的覆盖而消失。事实上，将图形或图片画在 TPainBox 对象上，同样面临这样的问题。但惟一的不同是，前者必须重画整个窗口，也就是 TForm 对象，而后者只要重画 TPainBox 对象即可。以下范例修改自范例 13-3，说明如何通过 TForm 对象的 onPaint 事件处理上述的问题。

范例 13-7

目的：本范例将示范如何通过 TForm 对象的 onPaint 事件重画 TCanvas 对象所绘制的图形或图片。

步骤 1 打开范例 13-3。

步骤 2 将 Button1 对象 onClick 事件处理过程的内容移动到 TForm 对象的 onPaint 事件处理过程，并删除 Button1 对象。

步骤 3 重新运行工程。工程运行后，系统会自动加载指定的图形文件。这是因为 TForm 对象被创建时，会自动调用 onPaint 事件。这个样的运行结果并不会因为窗口的重叠而消失。

范例 13-8

目的：本范例将示范如何通过 TPaintBox 对象的 onPaint 事件处理过程加载指定的图形文件，以处理画面的重画。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TPaintBox 对象，并在其 onPaint 事件处理过程加入以下的代码：

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  Bitmap : TBitmap;
begin
  try
    //创建一个 TBitmap 对象
    Bitmap := TBitmap.Create;
    //加载指定的图形文件
    Bitmap.LoadFromFile('FACTORY.BMP');
    //设置笔刷的样式
    PaintBox1.Canvas.Brush.Bitmap := Bitmap;
    //画出一个与图形文件同样大小的方形
    PaintBox1.Canvas.FillRect(Rect(0, 0 , Bitmap.Width, Bitmap.Height));
  finally
    //取消笔刷的样式
    PaintBox1.Canvas.Brush.Bitmap := nil;
    //释放 TBitmap 对象
    Bitmap.Free;
  end;
end;
```

步骤 3 保存运行工程。工程运行后，系统会自动加载指定的图形文件。这是因为 TForm 对象被创建时，会自动重画所有的组件。因此 TPaintBox 对象的 onPaint 事件自然就被调用了。这样的运行结果并不会因为窗口的重迭而消失。

注意 如果您比较范例 13-7 与 范例 13-8 的代码，您一定会发现为什么代码这么的像，也可以说是几乎完全相同。为什么呢?因为我们同样都是通过 TCanvas 对象来显示图片。由此可知，您可以使用 TCanvas 的功能于任何的 VCL 组件。

13.6 TBevel 对象

TBevel 对象位于对象面板的 Additional 页标签中，它与 TShape 对象有一点类似，它们都可以用于画面上的设置。惟一的不同，在于 TShape 对象可以拥有属于自己的背景颜色或事件，而 TBevel 对象则是一个透明的对象。TBevel 对象大部分的属性先前都已经介绍过，笔者这里只介绍属于 TBevel 对象的特有属性。

Style 属性

Style 属性决定此 TBevel 对象边框的样式，Style 属性有 bsRaised 与 bsLowered 两种属性值，如图 13.6 所示。

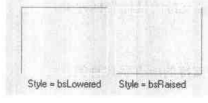


图 13.6 TBevel 对象的 Style 属性示范

Shape 属性

Shape 属性决定此 TBevel 对象所出现的边框，Shape 属性有 bsBox、bsFrame、bsTopLine、bsBottomLine、bsLeftLine、bsRightLine、bsSpacer 等属性值。当 TBevel 对象的 Shape 属性值为 bsSpacer 时，运行时期无法显示。如图 13.7 所示。

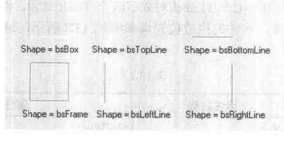


图 13.7 TBevel 对象的 Shape 属性示范

13.7 TScreen 对象

TScreen 对象用来描述目前应用程序所运行的屏幕环境，其被定义于 Forms 程序单元文件中。以下就 TScreen 对象常用的属性加以说明：

Width 与 Height 属性

Width 与 Height 属性可获取目前屏幕的宽度与高度，也就是所谓屏幕的分辨率。

- *WorkAreaHeight 与 WorkAreaWidth 属性*

WorkAreaHeight 与 WorkAreaWidth 属性可分别获取工作区域的高度与宽度。

- *WorkAreaLeft 与 WorkAreaTop 属性*

WorkAreaLeft 与 WorkAreaTop 属性可分别获取工作区域的左边界与上界坐标。

- *WorkAreaRect 属性*

WorkAreaRect 可获取工作区域的四点坐标值。

- *Cursors 属性*

Cursors 属性可获取目前屏幕支持的鼠标光标种类。

- *Fonts 属性*

Fonts 属性可获取目前屏幕支持的所有字体种类。

- *Imes 属性*

Imes 属性可获取目前屏幕支持的所有输入法种类。



13-9

目的： 本范例将示范如何使用 TScreen 对象常用的属性。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入七个 TLabel 对象、四个 TEdit 对象、两个 TComboBox 对象及一个 TMemo 对象，相关的摆放位置请参考图 13.8 所示。这些对象的相关属性设置如表 13.7 所示。

表 13.7

对象	属性名称	属性设置值
TLabel	Name	LabelWidth
	Caption	屏幕宽度
TLabel	Name	LabelHeight
	Caption	屏幕高度
TLabel	Name	LabelWorkAreaWidth
	Caption	工作区域宽度
TLabel	Name	LabelWorkAreaHeight
	Caption	工作区域高度
TLabel	Name	LabelIME
	Caption	所有输入法

续上表

对象	属性名称	属性设置值
TLabel	Name	LabelFont
	Caption	所有字体
TLabel	Name	LabelFontTesting
	Caption	Font Testing
TEdit	Name	EditWidth
TEdit	Name	EditHeight
TEdit	Name	EditWorkAreaWidth
TEdit	Name	EditWorkAreaHeight
TComboBox	Name	ComboBoxIME
TComboBox	Name	ComboBoxFont
TMemo	Name	Memo1

STEP 3 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码，这段代码主要是获取屏幕的宽度与高度、工作区域的宽度与高度、及所有的字体与输入法。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //获取屏幕的宽度与高度
  EditWidth.Text := IntToStr(Screen.Width);
  EditHeight.Text := IntToStr(Screen.Height);
  //获取工作区域的宽度与高度
  EditWorkAreaWidth.Text := IntToStr(Screen.WorkAreaWidth);
  EditWorkAreaHeight.Text := IntToStr(Screen.WorkAreaHeight);
  //获取所有输入法
  ComboBoxIME.Items := Screen.Imes;
  //获取所有字体
  ComboBoxFont.Items := Screen.Fonts;
end;
```

STEP 4 在 ComboBoxIME 对象的 onChange 事件处理过程加入以下的代码，这段代码主要是将输入焦点移到 Memo1 对象，并打开指定的输入法。

```
procedure TForm1.ComboBoxIMEChange(Sender: TObject);
begin
  //设置 Memo1 对象的输入法，以利用户输入
  Memo1.ImeName := ComboBoxIME.Items[ComboBoxIME.ItemIndex];
  //Memo1 对象获取焦点
  Memo1.SetFocus;
end;
```

STEP 5 在 ComboBoxFont 对象的 onChange 事件处理过程中加入以下的代码，这段代码主要是将用户所指定的输入法应用到 LabelFontTesting 对象。

```
procedure TForm1.ComboBoxFontChange(Sender: TObject);
begin
   //直接改变 LabelFontTesting 对象的字体
   if ComboBoxFont.ItemIndex <> -1 then
     LabelFontTesting.Font.Name :=
                  ComboBoxFont.Items[ComboBoxFont.ItemIndex];
end;
```

6 保存并运行工程。工程运行后，您可以查看目前屏幕及工作区域的大小与所有的字体和输入法。当您选择某个字体时，程序会将其应用于“Font Testing”的 TLabel 对象。另外，当您选择某个输入法时，程序会将焦点移至 TMemo 对象并打开指定的输入法，如图 13.8 所示。



图 13.8 TScreen 对象运行范例

13.8 TColorGrid 对象

TColorGrid 对象位于组件面板的 Samples 页标签中，其提供一个 16 个颜色的选择模板。TColorGrid 对象允许您分别使用鼠标左、右键设置前景与背景颜色。您也可以通过 ForegroundColor 与 BackgroundColor 属性分别获取您所设置的前景与背景颜色。

范例 13-10

目的：本范例将示范如何使用 TColorGrid 对象设置 TLabel 对象的前景与背景颜色。

1 创建新的工程。

2 在 TForm 对象中加入一个 TLabel 对象与一个 TColorGrid 对象，这些对象的相关属性设置如表 13.8 所示。

表 13.8

对象	属性名称	属性设置值
TLabel	Name	LabelColorTesting
	Caption	Color Testing
TColorGrid	Name	ColorGrid1

3 在 TColorGrid 对象的 onClick 事件处理过程中加入以下的代码，这些代码主要是获取 TColorGrid 对象所设置的前景与背景颜色，并将其应用于 LabelColorTesting 对象。

```
procedure TForm1.ColorGrid1Click(Sender: TObject);
begin
  //前景颜色
  LabelColorTesting.Font.Color := ColorGrid1.ForegroundColor;
  //背景颜色
  LabelColorTesting.Color := ColorGrid1.BackgroundColor;
end;
```

4 保存并运行工程。工程运行后，您可通过鼠标的左键与右键分别选择前景与背景颜色，此时程序会将您的选择应用于 LabelColorTesting 对象，如图 13.9 所示。

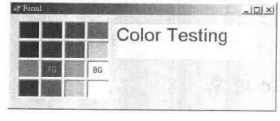


图 13.9　TColorGrid 对象运行范例

13.9　TColorBox 对象

TColorBox 对象位于组件面板的 Addtional 页标签中，它提供较多的颜色，但只能设置一个颜色。您可以通过 TColorBox 对象的 Selected 属性获取目前被选择的颜色。

范例 13-11

目的：本范例将示范如何使用 TColorBox 对象设置 TLabel 对象的字体颜色。

1 创建新的工程。

2 在 TForm 对象中加入一个 TColorBox 对象与 TLabel 对象，其相关设置如表 13.9 所示。

表 13.9

对象	属性名称	属性设置值
TLabel	Name	LabelColorTesting
	Caption	Color Testing
TColorBox	Name	ColorBox1

3 在 TColorBox 对象的 onChange 事件处理过程中加入以下的代码，这段代码主要的目的是获取 TColorBox 对象被选取的颜色，并将其应用于 TLabel 对象。

```
procedure TForm1.ColorBox1Change(Sender: TObject);
begin
  //设置字体颜色
```

```
    LabelColorTesting.Font.Color := ColorBox1.Selected;
end;
```

步骤 4 保存并运行工程。工程运行后，您可选择指定的颜色。此时程序会将您的选择应用于 LabelColorTesting 对象，如图 13.10 所示。

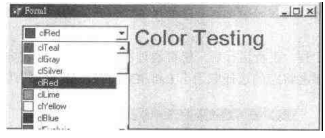


图 13.10 TColorBox 对象运行范例

13.10 TAnimate 对象

Delphi 6 所提供的多媒体组件包括 TAnimate 对象与 TMediaPlayer 对象。TAnimate 对象位于组件面板的 Win32 页标签中，它是负责处理没有声音的多媒体数据，而 TMediaPlayer 对象则负责声音与影片的多媒体数据。本小节将介绍如何使用 TAnimate 对象处理多媒体数据，下一个小节中再为您介绍 TMediaPlayer 对象。以下就 TAnimate 对象常用的属性、操作方法及事件处理加以说明。

- *Filename* 属性

FileName 属性用来指定多媒体(AVI)文件的保存位置。

- *Repetition* 属性

Repetition 属性用来决定多媒体(AVI)文件的播放的次数，如果次数为 0，那么该文件就会一直重复的播放，一直到调用 Stop 操作方法为止。

- *FrameCount* 属性

FrameCount 属性可获取目前多媒体(AVI)文件的帧数(Frame)。

- *StartFrame* 属性

StartFrame 属性用来决定多媒体(AVI)文件第一个被播放的帧(Frame)。

- *StopFrame* 属性

StopFrame 属性用来决定多媒体(AVI)文件最后一个被播放的帧(Frame)。

- *CommonAVI* 属性

CommonAVI 属性用来决定多媒体(AVI)的类型，如果您要播放指定的文件，那么

CommonAVI 属性值必须设为 aviNone，也就是表示不使用系统默认的 AVI。以下将以图标来说明系统默认的八种 AVI 类型，如图 13.11～图 13.18 所示：

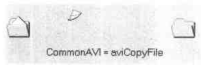


图 13.11　CommonAVI 属性值为 aviCopyFile

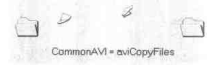


图 13.12　CommonAVI 属性值为 aviCopyFiles

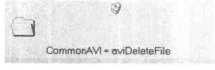


图 13.13　CommonAVI 属性值为 aviDeleteFile

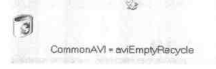


图 13.14　CommonAVI 属性值为 aviEmptyRecycle



图 13.15　CommonAVI 属性值为 aviRecycleFile



图 13.16　CommonAVI 属性值为 aviFindFolder



图 13.17　CommonAVI 属性值为 aviFindComputer



图 13.18　CommonAVI 属性值为 aviFindFile

- ***Play* 操作方法**

 Play 操作方法可用来开始播放多媒体(AVI)文件。

- ***Stop* 操作方法**

 Stop 操作方法可用来停止播放多媒体(AVI)文件。

- ***Reset* 操作方法**

 Reset 操作方法可重新设置被播放的多媒体(AVI)文件，也就是回到影片的第一帧(Frame)。

- ***Seek* 操作方法**

 Seek 操作方法可播放指定多媒体(AVI)文件中的特定帧(Frame)。

- ***onOpen* 事件**

 当 TAnimate 对象被打开时会触发 onOpen 事件。

- *onClose* 事件

 当 TAnimate 对象被关闭时会触发 onClose 事件。

- *onStart* 事件

 当 TAnimate 对象开始播放时会触发 onStart 事件。

- *onStop* 事件

 当 TAnimate 对象结束播放时会触发 onStop 事件。

范例 13-12

目的：本范例将示范如何使用 TAnimate 对象显示指定的 AVI 文件。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TAnimate 对象，并将其 FileName 属性值设为“C:\Program Files\Borland\Delphi6\Demos\ CoolStuf\cool.avi”。这是 Delphi 一个范例程序所附带的文件，可能会按您安装的目录不同而有所不同。最后再将其 Active 属性设为 True。
3. 保存并运行工程。工程运行后，这个 AVI 文件会一直被播放，那是因为 Repetition 属性值为 0 的缘故，如 13.19 所示。



图 13.19 Tanimate 对象运行范例

13.11 TMediaPlayer 对象

TMediaPlayer 对象负责处理声音与影片的多媒体数据，位于组件面板的 System 页标签中。它除了可以播放一般的 AVI 影片外，还可以播放音乐 CD，甚至还可以播放 MP3 的文件，真是太神奇了。事实上，笔者发现 TMediaPlayer 对象会根据不同的多媒体数据格式而打开系统对应的应用程序来播放。以 MP3 而言，TMediaPlayer 对象是调用 Windows Media Player 来播放的。以下就 TMediaPlayer 对象常用的属性与操作方法加以说明。

- *DeviceType* 属性

 DeviceType 属性可决定 TMediaPlayer 对象播放时所使用的设备种类。一般来说，如果 DeviceType 属性值为 dtAutoSelect 时，系统会根据 FileName 属性的扩展名自动决定所使用的设备名称。

- *FileName* 属性

FileName 属性用来指定多媒体文件的保存位置。

- *Display* 属性

Display 属性用来指定多媒体播放的对象，也就是说影片被播放时所出现的位置。通常我们会指定一个对象，例如:TPanel 等。

- *AutoOpen* 属性

当 AutoOpen 属性为 True 时，当 TMediaPlayer 对象被创建时则会自动打开。否则，必须调用 Open 操作方法来打开 TMediaPlayer 对象。

- *VisibleButtons* 属性

VisibleButtons 属性用来决定 TMediaPlayer 对象要出现哪些“按钮”，TMediaPlayer 对象所提供的按钮包括:btPlay、btPause、btStop、btNext、btPrev、btStep、btBack、btRecord 及 btEject 等。

- *EnabledButtons* 属性

EnabledButtons 属性用来决定 TMediaPlayer 对象的哪些“按钮”可以被使用。

- *AutoEnable* 属性

当 AutoEnable 属性为 True 时，TMediaPlayer 对象会自动控制“按钮”是否可以被使用。

- *Open* 操作方法

Open 操作方法可打开指定的多媒体设备。

- *Close* 操作方法

Close 操作方法可关闭指定的多媒体设备。

- *Play* 操作方法

Play 操作方法可开始播放指定的多媒体文件。

- *Stop* 操作方法

Stop 操作方法会停止播放指定的多媒体文件。

- *Pause* 操作方法

Pause 操作方法会暂停播放指定的多媒体文件。

- *Resume* 操作方法

Resume 操作方法会继续播放被暂停的多媒体文件。

范例 13-13

目的：本范例将示范如何使用 TMediaPlayer 对象播放指定的多媒体文件。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TMediaPlayer 对象，并将其 FileName 属性值设为 C:\ProgramFiles\Borland\Delphi6\ Demos\CoolStuf\ speedis.avi。这是 Delphi 一个范例程序所附带的文件，可能会依您安装的目录不同而有所不同。最后再将其 AutoOpen 属性设为 True。
3. 在 TForm 对象中另外加入一个 TPanel 对象，其默认的名称为 Panel1，并将 TMediaPlayer 对象的 Display 属性值设为 Panel1。
4. 保存并运行工程。工程运行后，此时指定的多媒体设备已经被打开了，您只要单击“Play”按钮，影片便立即播放，如图 13.20 所示。



图 13.20 TMediaPlayer 对象运行范例

第14章

Delphi 6 Action管理对象

本章目标

本章将为大家介绍Delphi 6所提供的操作(Action)相关对象。TActionList对象允许您将一些常用的命令或操作加以集中管理，而每一个命令或操作就称为一个 Action。除此之外，还要为您介绍 TActionManager 对象、TActionMainMenu 对象、TActionToolBar 对象及 TCustomizeDlg 对象等。

本章主题

14.1 TactionList 对象

14.2 自定义 TAction 对象

14.3 TActionManager 对象

14.4 TActionMainMenu 对象

14.5 TActionToolBar 对象

14.6 TCustomizeDlg 对象

14.1 TActionList 对象

TActionList 对象是 Delphi 6 众多对象中一个相当特别的对象，您可以称它为“操作列表”对象。以下就“操作(Action)”的相关概念详加说明：

- 操作列表对象：也就是 TActionList 对象。操作列表对象允许您将一些常用的命令或操作加以集中管理，这些命令或操作我们称它为“操作(Action)”。
- 操作对象：也就是 TAction 对象。TAction 对象是操作的实例，例如标准的编辑操作对象就包括：TEditCopy、TEditCut、TEditDelete、TEditPaste、TEditSelectAll 及 TEditUndo 等。
- 操作链接对象：也就是 TActionLink 对象。TActionLink 对象是负责维护操作与用户对象之间的链接状况，它也决定用户对象是否允许被使用指定的操作对象。
- 用户对象：每一个操作必须对应于一个用户对象，这种用户对象通常会是一个按钮(TButton、TSpeedButton、TToolButton、TRadioButton)或菜单(TMenuItem)，而且这些用户对象的 onClick 事件将会被对应于 TAction 的 onExecute 事件处理过程。
- 操作的目标对象：有些“操作”需要对应一个目标对象，它通常也是一个 VCL 组件，例如：TMemo、TRichEdit 或数据集(DataSet)对象，如图 14.1 所示。



图 14.1 TActionList 对象图例说明

图 14.1 中说明操作列表对象、操作对象、操作链接对象、用户对象及目标对象之间的关系。其中 Paste 的 TMenuItem 对象通过操作链接对象，链接到 TActionList 对象中的 TEditPaste 操作对象，而 Memo1 则是此操作的用户对象。

TActionList 对象将一些常用的功能封装在标准操作(Standard Action)中，这些操作对象可分为标准的编辑操作(Edit)、文字格式化操作(Format)、联机帮助(Help)、标准的窗口操作(Window)、文件处理(File)、搜索操作(Search)、页标签操作(Tab)、列表对象的操作(List)、对话框的操作(Dialog)、因特网操作(Internet)、数据集(DataSet)相关操作及工具(Tool)等 12 大类，详细如下：

- 标准的编辑操作(Edit)包括：EditCopy、EditCut、EditDelete、EditPaste、EditSelectAll 及 EditUndo 等。
- 文字格式化操作(Format)包括：TRichEditBold、TRichEditItalic、TRichEditUnderline、TrichEdit StrikeOut、TRichEditBullets、TRichEditAlignLeft、TRichEditAlignRight、TRichEditCenter 等。

- 联机帮助(Help)包括：HelpContents、HelpOnHelp、HelpTopicSearch 及 HelpTopicSearch 等。
- 标准的窗口操作(Window)包括：WindowArrange、WindowCascade、WindowClose、WindowMinimizeAll、WindowTileHorizontal 及 WindowTileVertical 等
- 文件处理(File)包括：TFileOpen、TFileSaveAs、TFilePrintSetup、TFileRun、TFileExit 等。
- 搜索操作(Search)包括：TSearchFind、TSearchFindNext、TSearchReplace、TSearchFindFirst 等。
- 页标签操作(Tab)包括：TPreviousTab、TNextTab 等。
- 列表对象的操作(List)包括：TListControlCopySelection、TListControlDeleteSelection、TListControlSelectAll、TListControlClearSelection、TlistControlMove Selection、TStaticListAction、TVirtualListAction 等。
- 对话框的操作(Dialog)包括：TOpenPicture、Tsave Picture、TColorSelect、TFontEdit、TPrintDlg 等。
- 因特网操作(Internet)包括：TBrowserURL、Tdownload URL、TSendMail 等。
- 数据集(DataSet)相关操作包括：TDataSetCancel、TDataSetDelete、TDataSetEdit、TDataSetFirst、TDataSetInsert、TDataSetLast、TDataSetNext、TDataSetPost、TDataSetPrior 及 TDataSetRefresh 等。
- 工具(Tool)包括附加工具如 TCustomizeActionBars 对象等。

以下将针对 TActionList 对象常用的操作，通过实例的方式，说明如何简化程序的开发。

范例 14-1

目的：本范例将示范如何使用 TActionList 对象文字处理的相关操作，如编辑操作(Edit)、文字格式化操作(Format)及搜索操作(Search)。

步骤 1 创建新的工程。

步骤 2 在工程中加入一个 TMainMenu 对象、一个 TRichEdit 对象及一个 TActionList 对象。

步骤 3 为 TActionList 对象加入 Edit、Format 及 Search 等三大类所有的操作。请双击“TActionList”对象，并从“操作编辑”窗口中加入 Edit、Format 及 Search 等类的所有操作，如图 14.2 所示。

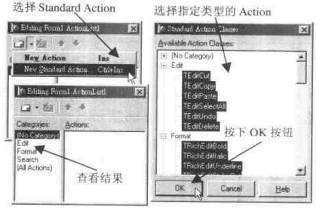


图 14.2　加入指定的操作项目

STEP 4 为 TMainMenu 对象加入“编辑”、“格式”及“搜索”等三个功能选项，如图 14.3 所示。



图 14.3 加入指定的功能类别

STEP 5 将 TActionList 对象中的每一个操作加入 TMainMenu 对象对应的分类中，以图 14.4 “Undo”为例。



图 14.4 将操作加入 TMainMenu 对象

STEP 6 重复上一个步骤的示范，将 TActionList 对象中的每一个操作加入 TMainMenu 对象对应的分类，如图 14.5 所示。

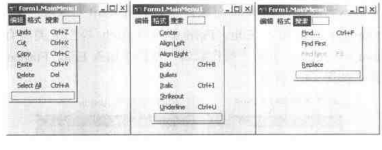


图 14.5 将所有的操作加入 TMainMenu 对象

STEP 7 保存并运行工程。工程运行后，您可以使用任何的文字相关功能，无论是文字本身的改变、搜索或是替换的功能，如图 14.6 所示。



图 14.6 TActionList 对象运行范例

范例 14-2

目的：本范例将示范如何使用 TActionList 对象与文件处理有关的操作，如打开、另存为、打印机设置、关闭应用程序，以及运行一个外部文件或程序。

步骤 1 打开范例 14-1。

步骤 2 为 TActionList 对象加入文件类型(File)的所有操作选项。

步骤 3 将上一步骤加入的操作选项添加至 TMainMenu 对象中，如图 14.7 所示。



图 14.7　将新的操作加入 TMainMenu 对象

步骤 4 由于这些文件的操作选项都会对应一个对话框，例如：打开、另存为或是打印机设置等。您可以从对象检查器中修改这些对话框相关的属性，如图 14.8 所示，这里以“打开”为例。



图 14.8　设置对话框的相关属性

步骤 5 使用“打开”与“另存为”的操作选项时，您必须特别注意，虽然系统可以自动打开指定的对话框，也可以获取指定的文件名称。但其后续的处理操作，还是需要您自行编写，您可以将这些代码加入操作选项的 onAccept 事件处理过程中，如图 14.9 所示。



图 14.9　设置打开文件的后续处理操作

步骤 6 参考上一个步骤为“打开”与“另存为”等操作选项的 onAccept 事件处理过程加入以下的代码：

```
procedure TForm1.FileOpen1Accept(Sender: TObject);
begin
  //将指定的文件内容加载 RichEdit1 对象
  RichEdit1.Lines.LoadFromFile(FileOpen1.Dialog.FileName);
end;

procedure TForm1.FileSaveAs1Accept(Sender: TObject);
begin
  //将 RichEdit1 对象的内容存到指定的文件中
  RichEdit1.Lines.SaveToFile(FileSaveAs1.Dialog.FileName);
end;
```

步骤 7 使用“运行(Run)”操作选项时，您必须特别注意 TFileRun 对象的几个属性值。Browse 属性代表是否允许用户在运行时期选择所要运行的外部文件、FileName 属性代表指定的外部文件名称，您可以在设计时期设置其属性值、Parameters 属性则代表运行此外部文件或应用程序所需要的参数，如图 14.10 所示。

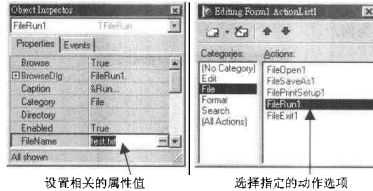


图 14.10 设置操作对象的相关属性

步骤 8 保存并运行工程。工程运行后，您可以打开指定的文本文件，或是将 RichEdit1 对象的内容保存到指定的文本文件中，也可以打开一个外部文件或运行一个外部应用程序。

范例 14-4

目的：本范例将示范如何使用 TActionList 对象与页标签(Tab)有关的操作，如上一个页标签或下一个页标签。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TPageControl 对象并添加五个页标签，也就是 TTabSheet 对象，如图 14.11 所示。

步骤 3 另外加入两个 TButton 对象及一个 TActionList 对象，并为 TActionList 对象加入与

页标签有关的操作。

4. 将 PreviousTab1 与 NextTab1 对象的 TabControl 属性设置为 PageControl1。
5. 分别将 PreviousTab1 与 NextTab1 等操作加入上一步骤所创建的 TButton 对象。
6. 保存并运行工程。工程运行后，您可以通过两个 TButton 对象轻易的切换页标签，如图 14.11 所示。

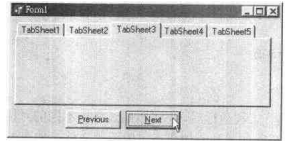


图 14.11　页标签操作范例

范例 14-4

目的：本范例将示范如何使用 TActionList 对象与对话框(Dialog)有关的操作，如 TOpenPicture、TSavePicture、TColorSelect、TFontEdit、TPrintDlg 等。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TActionList 对象并加入与对话框有关的操作选项。
3. 另外加入一个 TMainMenu 对象、TLabel 对象与 TImage 对象。
4. 将步骤 2 所加入的操作选项加入 TMainMenu 对象中。
5. 分别为 OpenPicture1、SavePicture1、ColorSelect1、FontEdit1 与 PrintDlg1 等操作选项对象的 onAccept 事件处理过程加入以下的代码：

```
procedure TForm1.OpenPicture1Accept(Sender: TObject);
begin
  //打开指定的图片文件
  Image1.Picture.LoadFromFile(OpenPicture1.Dialog.FileName)
end;

procedure TForm1.SavePicture1Accept(Sender: TObject);
begin
  //保存指定的图片文件
  Image1.Picture.SaveToFile(SavePicture1.Dialog.FileName)
end;

procedure TForm1.ColorSelect1Accept(Sender: TObject);
begin
   //设置指定的颜色
   Label1.Font.Color := ColorSelect1.Dialog.Color;
```

```
end;
procedure TForm1.FontEdit1Accept(Sender: TObject);
begin
   //设置指定的字体
   Label1.Font := FontEdit1.Dialog.Font;
end;
procedure TForm1.PrintDlg1Accept(Sender: TObject);
begin
   //打印出 Form1 的画面
   Form1.Print;
end;
```

步骤 6 保存并运行工程。工程运行后，您可以通过各种不同的对话框分别为 TLabel 对象或 TImage 对象改变字体属性或加载指定的图片文件，如图 14.12 所示。



图 14.12 对话框操作范例

范例 14-5

目的：本范例示范如何使用 TActionList 对象与因特网(Internet)有关的操作，如 TBrowserURL、TDownloadURL、TSendMail 等。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TActionList 对象并加入与因特网有关的操作项目。

步骤 3 另外加入一个 TMainMenu 对象并将上一个步骤所加入的操作项目加入 TMainMenu 对象。

步骤 4 使用 TBrowserURL 与 TDownloadURL 操作项目时，您必须特别注意其 URL 属性值，请设置您要浏览或下载的网址。

步骤 5 保存并运行工程。工程运行后，您可以浏览、下载指定的网址。当然，您也可以通过 TSendMail 发送电子邮件。如果您要发送电子邮件，系统打开默认的电子邮件工具，以图 14.13Outlook 为例。

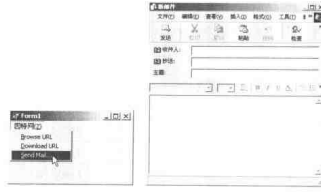


图 14.13　因特网相关操作项目范例

范例 14-6

目的：本范例将示范如何使用 TActionList 对象与数据集(DataSet)有关的操作，本范例以 TDataSetNext 与 TDataSetPrior 为例。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入 TADOConnection、TADOTable、TDataSource 及 TDBGrid 等对象。有关数据库的连接请参考第 20 章的详细说明。

STEP 3 另外在 TForm 对象中加入一个 TActionList 对象并加入与数据集有关的操作选项，并设置所有操作选项的 DataSource 属性值为 DataSource1。

STEP 4 在 TForm 对象中加入两个 TButton 对象，并将 TActionList 对象的 TDataSetNext 与 TDataSetPrior 等操作选项分别指定给这两个 TButton 对象。

STEP 5 保存并运行工程。工程运行后，您可以通过上一个步骤所创建的两个 TButton 对象来改变数据集的光标，如图 14.14 所示。

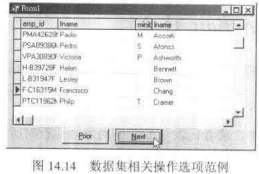


图 14.14　数据集相关操作选项范例

范例 14-7

目的：本范例将示范如何使用 TActionList 对象与列表对象(ListView)有关的操作，本范例以 TListControlCopySelection、TListControlDeleteSelection、TlistControlClearSelection 与 TListControlMoveSelection 为例。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入两个 TListView 对象，其相关属性设置如表 14.1 所示。

表 14.1

对象名称	属性名称	属性设置值
TListView	Name	ListView1
	ViewStyle	vsList
	MultiSelect	True
TListView	Name	ListView2
	ViewStyle	vsList
	MultiSelect	True

STEP 3 为 ListView1 对象的 Items 属性加入三个数据项，其标题文字分别为“第一个选项”、“第二个选项”及“第三个选项”等。

STEP 4 在 TForm 对象另外加入一个 TMainMenu 对象与一个 TActionList 对象。并且为 TActionList 对象加入与列表对象有关的操作选项，并将所有操作选项对象的 ListControl 属性值设为 ListView1，并且将 Destination 属性值设为 ListView2。

STEP 5 保存并运行工程。工程运行后，您可以通过列表对象的操作功能，将指定的数据项移动或复制到 ListView2 对象，如图 14.15 所示。



图 14.15 列表对象相关的操作选项范例

14.2 自定义 TAction 对象

TActionList 对象除了将一些常用的功能封装在 Standard Action 中，它还允许我们使用自定义的 TAction 对象，本小节将示范如何自定义 TAction 对象。在示范如何自定义 TAction 对象之前，我们先来谈谈一个 TAction 对象包含哪些事件。每一个 TAction 对象都有 onExecute 及 onUpdate 两个重要事件。前者代表 TAction 对象被运行时所触发的事件，而后者则代表 TAction 对象被更新时所触发的事件。这样解释您也许不太明白，以下将以实例来说明如何自定义 TAction 对象。

范例 14-8

目的：本范例将示范如何自定义 TAction 对象。

STEP 1 创建新的工程。

(2) 在工程中加入一个 TButton 对象、一个 TEdit 对象及一个 TActionList 对象。

(3) 为 TActionList 对象加入一个 New Action，此 Action 的名称默认为 Action1，如图 14.16 所示。

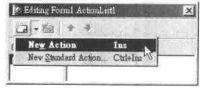


图 14.16　创建自定义的 Action

(4) Action1 的相关属性设置如表 14.2 所示。

表 14.2

对象名称	属性名称	属性设置值
Action1	Caption	保存
	ShortCut	Ctrl+S

(5) 在 Action1 的 onExecute 事件处理过程中加入以下的代码：

```
procedure TForm1.Action1Execute(Sender: TObject);
begin
  Showmessage('您按下 TButton 对象');
end;
```

(6) 在 Action1 的 onUpdate 事件处理过程中加入以下的代码：

```
procedure TForm1.Action1Update(Sender: TObject);
begin
  Button1.Enabled := Edit1.Modified;
end;
```

(7) 将 TButton 对象的 Action 属性设置为 Action1。此时 TButton 对象的 Caption 属性立即自动被更改为 Action1 的 Caption 属性。

(8) 保存工程。

(9) 按【F9】键运行工程，工程运行时，TButton 对象的 Enabled 属性会自动被设为 False。但是当 TEdit 对象的内容被更改时，TButton 对象的 Enabled 属性即自动被设为 True，如图 14.17 所示。

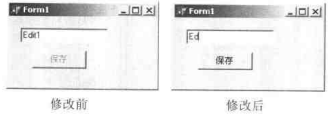


图 14.17　自定义 TAction 运行结果

14.3 TActionManager 对象

TActionManager 对象位于组件面板的 Additional 页标签中，它是一个操作(Action)的管理组件。通常 TActionManager 对象会配合 TActionMainMenu 对象、TActionToolBar 对象或 TCustomizeDlg 对象一起使用。其中 TActionMainMenu 对象就像 TMainMenu 一样，但它是专门为操作(Action)而设计的，TActionToolBar 对象同样是为了操作(Action)而设计的，它与 TToolBar 对象在用法上是相同的。

如果您在 TActionManager 对象上快按两下鼠标，此时会打开一个“TActionManager”对象的编辑对话框。在这个对话框中有“ToolBars”、“Actions”及“Options”等三个页标签。

- “ToolBars”页标签：主要是用来添加或删除 TActionManager 对象所属的 TActionToolBar 对象，如图 14.18 所示。
- “Actions”页标签：主要是用来设置 TActionManager 对象所包含的操作，它就像是一个 TActionList 对象，如图 14.19 所示。
- “Options”页标签：主要用来设置一些关于 TActionMainMenu 对象与 TActionToolBar 对象的可视化选项，如图 14.20 所示。



图 14.18 TActionManager 对象的“ToolBars”页标签

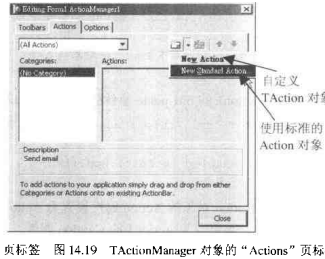


图 14.19 TActionManager 对象的“Actions”页标签

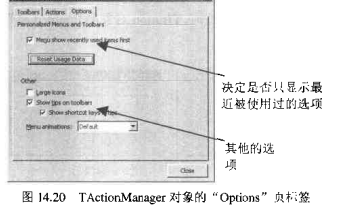


图 14.20 TActionManager 对象的“Options”页标签

14.4　TActionMainMenu 对象

TActionMainMenu 对象是一种专门为操作(Action)所设计的下拉式菜单，它的用法非常类似于 TMainMenu 对象。以下范例将示范如何为 TActionMainMenu 对象加入 TActionManager 对象所提供的标准操作(Standard Action)。

范例 14-9

目的：本范例将示范如何为 TActionMainMenu 对象加入 TActionManager 对象所提供的标准操作。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象加入一个 TActionManager 对象与 TActionMainMenu 对象，并将 TActionMainMenu 对象的 ActionManager 属性值设为“ActionManager1”。

步骤 3　另外为 TForm 对象中加入一个 TActionManager 对象，并为其加入“File”类别的标准操作。

步骤 4　通过拖动的方式将 File 类别加入 TActionMainMenu 对象，如图 14.21 所示。

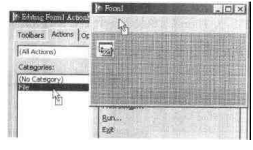


图 14.21　为 TActionMainMenu 对象加入 File 类型的操作

步骤 5　请查看 TActionMainMenu 对象所加入的 File 类型操作，此时系统已经自动将指定的操作加入 TActionMainMenu 对象，如图 14.22 所示。



图 14.22　查看 TActionMainMenu 对象的 File 类型的操作

注意　与 TMainMenu 对象相同，TActionMainMenu 对象与 TActionToolBar 对象同样可以配合 TImageList 对象一起使用。有关 TImageList 对象的使用方法，请参考第 13 章的说明。

14.5 TActionToolBar 对象

TActionToolBar 对象也是一种专门为操作(Action)所设计的工具栏对象，它的用法非常类似于 TToolBar 对象。以下范例将示范如何为 TActionToolBar 对象加入 TActionManager 对象所提供的标准操作(Standard Action)。

范例 14-10

目的： 本范例将示范如何为 TActionToolBar 对象加入 TActionManager 对象所提供的标准操作。

步骤 1 打开范例 14-9。

步骤 2 在 TActionManager 对象上双击，此时会打开一个“TActionManager”对象的编辑对话框。并且通过此对话框为 TForm 对象加入一个 TActionToolBar 对象，如图 14.23 所示。



图 14.23 为 TForm 对象加入一个 TActionToolBar 对象

步骤 3 通过拖动的方式将 File 类型加到 TActionToolBar 对象，如图 14.24 所示。



图 14.24 为 TActionToolBar 对象加入 File 类型的操作

步骤 4 请查看 TActionToolBar 对象所加入的 File 类型操作，此时系统已经自动将指定的操作加入 TActionToolBar 对象，如图 14.25 所示。



图 14.25 查看 TActionToolBar 对象的 File 类型的操作

14.6　TCustomizeDlg 对象

TCustomizeDlg 对象也是一种专门为操作(Action)所设计的对象，它允许用户在运行时期自行调整 TActionMainMenu 对象与 TActionToolBar 对象的视觉效果。以下范例将示范如何为 TActionMainMenu 对象加入 TCustomizeDlg 对象。



14-11

目的：本范例将示范如何为 TActionToolBar 对象加入 TActionManager 对象所提供的标准操作。

1. 打开范例 14-10。
2. 在 TForm 对象上加入一个 TCustomizeDlg 对象，并且将其 ActionManager 属性值设为“ActionManager1”。
3. 在 TActionManager 对象上双击，此时会打开一个“TActionManager”对象的编辑对话框。通过此对话框为 TActionManager 对象加入 Tool 类型的操作。
4. 通过拖动的方式将 Tool 类型加到 TActionMainMenu 对象。
5. 保存并运行工程。工程运行后，您可通过菜单栏中的“Tool/Customize”选项打开运行时期的 TCustomizeDlg 对象，如图 14.26 所示。

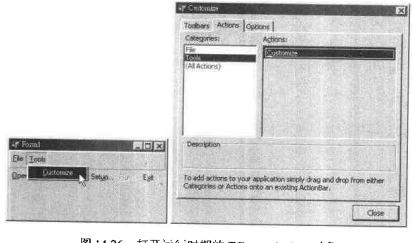


图 14.26　打开运行时期的 TCustomizeDlg 对象

第 15 章

Delphi 6 其他常用对象

本章目标

本章将延续前面的章节继续讨论 Delphi 6 所提供的其他 13 个常用对象，这些对象包括可视化组件与非可视化组件，请参考各小节的详细说明。

本章主题

15.1 TscrollBar 对象

15.2 TProgressBar 对象

15.3 TStatusBar 对象

15.4 TTimer 对象

15.5 TSplitter 对象

15.6 TDateTiemPicker 对象

15.7 TMonthCalendar 对象

15.8 TFrame 对象

15.9 TApplication 对象

15.10 TRegistry 对象

15.11 TIniFile 对象

15.12 TList 对象

15.13 TStrings 与 TStringList 对象

15.1 TScrollBar 对象

本小节主要说明 TScrollBar 对象，此对象位于组件面板的 Standard 页标签上，最常被使用于滚动画面。在 Delphi 6 中也有不少对象提供类似滚动的属性，例如 TList、TStringGrid 对象等。以下就 TScrollBar 对象经常被使用的属性加以说明。

- *Kind 属性*

Kind 属性是用来设置 TScrollBar 对象外观类型的属性，Kind 属性有 sbHorizontal 与 sbVertical 两种可能值。前者代表滚动条是水平滚动条，而后者则代表滚动条为垂直滚动条。

- *Max 属性*

Max 属性设置滚动条滚动范围的最大值。

- *Min 属性*

Min 属性设置滚动条滚动范围的最小值。

- *Position 属性*

Position 属性显示目前滚动条所在的位置。

- *LargeChange 属性*

LargeChange 属性设置鼠标每次点在滚动条上的滚动量。

- *SmallChange 属性*

SmallChange 属性设置鼠标点在滚动条方向按钮的滚动量。

范例 15-1

目的：本范例主要说明 TScrollBar 对象的 Max、Position 等各项属性的使用方法。

1. 创建新的工程。
2. 在 TForm 对象中加入两个 TScrollBar 对象，三个 TButton 对象，四个 TLabel 对象，四个 TEdit 对象，其属性值设置如表 15.1 所示。

表 15.1

对象	属性名称	属性设置值
TScrollbar	Name	HScrollbar
	Max	100
	Min	0
	Position	0
	Kind	SbHorizontal

续上表

对象	属性名称	属性设置值
TScrollbar	Name	VScrollbar
	Max	100
	Min	0
	Position	0
	Kind	SbVertical
TLabel	Name	Label1
	Caption	水平滚动条现在位置
TLabel	Name	Label2
	Caption	水平滚动最大范围
TLabel	Name	Label3
	Caption	垂直滚动条现在位置
TLabel	Name	Label4
	Caption	垂直滚动最大范围
TEdit	Name	HCurEdit
	Caption	0
TEdit	Name	HMaxEdit
	Caption	100
TEdit	Name	VCurEdit
	Caption	0
TEdit	Name	VMaxEdit
	Caption	100
TButton	Name	ButtonApply
	Caption	应用
TButton	Name	ButtonDefault
	Caption	默认值
TButton	Name	Button3
	Caption	Button3

步骤 3　在 Var 之后声明两个变量来存放滚动条之前的位置，如下：

```
var
  Form1: TForm1;
  //存放水平滚动条被改变前的位置
  HorizontalPosition : integer;
  //存放垂直滚动条被改变前的位置
  VerticalPosition : integer;
```

步骤 4　在 HScrollBar 对象的 OnChange 事件中填入以下代码：

```
procedure TForm1.HScrollBarChange(Sender: TObject);
begin
  //显示水平滚动条现在位置
  HCurEdit.text:= intToStr(HScrollBar.position);

  //并计算 Button3 要移动的点数(Pixel)
  if (HScrollBar.position-HorizontalPosition) > 0  then
    begin
      Button3.Left := Button3.Left - (HScrollBar.Position -
                                HorizontalPosition);
      HorizontalPosition:= HScrollBar.Position;
    end
  else if (HScrollBar.Position - HorizontalPosition ) < 0 then
    begin
      Button3.Left := Button3.Left - (HScrollBar.Position -
                                HorizontalPosition);
      HorizontalPosition:= HScrollBar.Position;
    end;
end;
```

步骤 5 在 VScrollBar 对象的 OnChange 事件中填入以下代码：

```
procedure TForm1.VScrollBarChange(Sender: TObject);
begin
  //显示垂直滚动条现在位置
  VCurEdit.text:= intToStr(VScrollBar.position);

  //计算 Button 3 要移动的点数(Pixel)
  if (VScrollBar.position-VerticalPosition) > 0  then
    begin
      Button3.Top := Button3.Top - (VScrollBar.Position -
                               VerticalPosition);
      VerticalPosition:= VScrollBar.Position;
    end
  else if (VScrollBar.Position - VerticalPosition ) < 0 then
    begin
      Button3.top := Button3.Top - (VScrollBar.Position -
                               VerticalPosition);
      VerticalPosition:= VScrollBar.Position;
    end;
end;
```

步骤 6 在 ButtonApply 对象的 OnClick 事件中填入：

```
procedure TForm1.ButtonApplyClick(Sender: TObject);
begin
  //取得 Button3 将被设置的水平及垂直的坐标位置
```

```
    HorizontalPosition:= HScrollBar.Position ;
    VerticalPosition:= VScrollBar.Position ;

    //设置 Button3 的新位置
    button3.left := HorizontalPosition + 80 ;
    button3.top  := VerticalPosition + 80 ;

    //设置水平滚动条
    HScrollBar.Position:=StrToInt(HCurEdit.text);
    HScrollBar.max:=StrToInt(HMaxEdit.text);

    //设置垂直滚动条
    VScrollBar.Position:=StrToInt(VCurEdit.text);
    VScrollBar.max:=StrToInt(VMaxEdit.text);
end;
```

步骤 7　在 ButtonDefault 的 OnClick 事件中填入：

```
procedure TForm1.ButtonDefaultClick(Sender: TObject);
begin
    // 设置所有属性为默认值
    HorizontalPosition:= 0;
    VerticalPosition:= 0;

    //设置 Button3 的位置
    button3.left:= 80;
    button3.top := 80;

    //设置水平滚动条
    HScrollBar.Position:=0;
    HScrollBar.max:=100;

    //设置垂直滚动条
    VScrollBar.Position:=0;
    VScrollBar.max:=100;

    //设置显示水平滚动条属性值的 TEdit 对象
    HCurEdit.text:= intToStr(HScrollBar.position);
    HMaxEdit.text:= intToStr(HScrollBar.max);

    //设置显示垂直滚动条属性值的 TEdit 对象
    VCurEdit.text:= intToStr(VScrollBar.position);
    VMaxEdit.text:= intToStr(VScrollBar.max);
end;
```

步骤 8　保存工程并按【F9】功能键运行工程。工程运行后，当水平或垂直滚动条被滚动时

会显示滚动条目前的位置，并移动 Button3 按钮。若要更改滚动条可滚动的最大值，需先输入数字再按下“应用”按钮。若按下“默认值”按钮，则会将水平与垂直滚动条的最大值与现在位置设为默认值并将 Button3 放到原来位置，如图 15.1 所示。

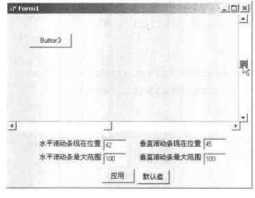


图 15.1 TScrollBar 运行结果

15.2 TProgressPar 对象

在这个小节中，将为您介绍 TProgressBar 对象，此对象位于 Win32 的组件面板上。TProgressBar 对象通常用于显示工作完成的进度比率。例如保存文件或安装程序时会出现在窗口下方的运行进度条。以下就 TProgressBar 对象经常被使用的属性加以说明。

- *Align* 属性

设置 TProgressBar 对象放置位置的属性，通常我们会将属性值设为 alBottom，使其出现在画面下方。

- *Max* 属性

Max 属性设置 TProgressBar 运行范围的最大值。

- *Min* 属性

Min 属性设置 TProgressBar 运行范围的最小值。

- *Step* 属性

Step 属性用来设置 TProgressBar 对象重复运行的次数。

- *StepIt* 操作方法

StepIt 操作方法用来启动 TProgressBar 对象。

范例 15-2

目的：本范例主要通过运行时期动态产生 TProgressBar 对象，以说明 TProgressBar 对象各项

属性的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TButton 对象、两个 TLabel 对象及两个 TEdit 对象，其相关属性设置如表 15.2 所示。

表 15.2

对象	属性名称	属性设置值
TButton	Name	Button
	Caption	确定
TLable	Name	Label1
	Caption	TProgressBar 最大值
TLable	Name	Label2
	Caption	TProgressBar 重复次数
TEdit	Name	EditTotal
TEdit	Name	EditStep

步骤 3 因为我们要在运行时期动态产生 TProgressBar 对象，因此您必须在 uses 之后引用 ComCtrls 单元。

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,  StdCtrls,ComCtrls;
```

步骤 4 我们在 Unit1 中创建一个自定义过程(Procedure)。这个自定义过程有 Total 与 StepQ 两个参数。前者代表 TProgressBar 对象的最大值(Max 属性)，后者代表 TProgressBar 重复运行的次数。

```
//声明
  TForm1 = class(TForm)
    procedure RunProgress(Total,StepQ: Integer);

//过程(Procedure)的内容
procedure TForm1.RunProgress(Total,StepQ: Integer);
var
  ProgressBar1: TProgressBar;
  i: Integer;
begin
  ProgressBar1 := TProgressBar.Create(Self);
  try
    ProgressBar1.Parent := Self;
    ProgressBar1.Align := alBottom;
    ProgressBar1.Min := 0;
    ProgressBar1.Max := Total;
    ProgressBar1.Step := StepQ; //重复的次数
```

```
    for i := 1 to Total do ProgressBar1.Stepit;
    ShowMessage('运行完毕!!');
  finally
    ProgressBar1.Free; //将 TProgressBar 对象 Free
  end;
end;
```

5 在 Button1 的 OnClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   //调用 RunProgress
  RunProgress(StrToInt(EditTotal.Text),StrToInt(EditStep.Text));
end;
```

6 保存工程并按下【F9】功能键运行工程。工程运行后，当用户单击“确定”按钮，系统会自动产生一个 TProgress 对象并显示运行进度，如图 15.2 所示。当运行完毕后会出现“运行完毕”的信息对话框，当用户单击“OK”按钮系统会自动释放 TProgress 对象。



图 15.2 运行 TProgressBar 对象

15.3 TStatusBar 对象

在本小节中我们将要讨论 TStatusBar 对象，此对象也是位于 Win32 的组件面板上，通常用于显示系统目前的状态信息。例如 Windows 98/NT/2000 中“我的电脑”窗口的状态栏就是一个典型的例子。如图 15.3 所示。



TStatusBar 对象的应用

图 15.3 TStatusBar 对象的应用

以下就 TStatusBar 对象经常被使用的属性加以说明。

Panels 属性

可将 TStatusBar 对象设置成类似 Microsoft word 的状态栏，有多项分隔的效果。图 15.4 是 Panels 属性的设置方法。

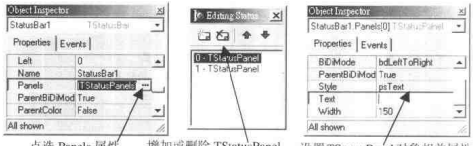


图 15.4　Panels 属性设置方法

SimplePanel 属性

SimplePanel 属性是设置 TStatusBar 显示简单状态栏。其属性值若为 True，则会显示 SimplePanel 状态栏。若为 False 则会显示 Panels 上所设置的状态栏。

SimpleText 属性

SimpleText 属性是设置 TStatusBar 上所会显示的信息。

范例 15-3

目的：本范例主要说明 TStatusBar 对象各项属性的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TStatusBar 对象，并添加两个 TStatusPanel 对象，其相关属性设置如表 15.3 所示。

表 15.3

对象	属性名称	属性设置值
TStatusBar	Name	StatusBar1
TStatusBar.Panels[0]	Width	150
TStatusBar.Panels[1]	Width	150

步骤 3 在 TForm1 的 OnMouseMove 事件中填入：

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
begin
```

```
  //当移动鼠标，系统会在第一区的状态栏上显示鼠标现在位置
  StatusBar1.Panels[0].Text := Format('鼠标现在位置：(%d, %d)',
                                      [X, Y]);
end;
```

步骤 4 在 TForm1 的 OnMouseDown 事件中填入：

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
  //当按下鼠标，系统会在第二区的状态栏上显示鼠标现在位置
  StatusBar1.Panels[1].Text := Format('鼠标按下的位置：
                              (%d, %d)', [X, Y]);
end;
```

步骤 5 保存工程，并按【F9】键运行。当您移动鼠标时，在窗口下方的第一区状态栏会出现鼠标现在的位置，当您按下鼠标时，会在第二区状态栏出现按下鼠标时的位置，如图 15.5 所示。

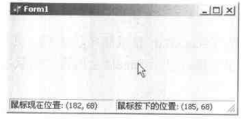


图 15.5 按下 Button1 所会出现的情况

范例 15-4

目的：本范例主要是说明如何将 TProgressBar 对象展示在 TStatusBar 对象中。

步骤 1 打开范例 15-2。

步骤 2 在 TForm 对象上加入一个 TStatusBar 对象，其相关属性的设置请参考范例 15-3。

步骤 3 在 TForm 对象上加入一个 TProgressBar 对象，并将其 Visible 属性值设为 False。

步骤 4 将范例 15-2 步骤 4 所创建的 RunProgress 过程内容改为下列代码：

```
procedure TForm1.RunProgress(Total,StepQ: Integer);
var
  i: Integer;
begin
  try
    //指定 ProgressBar1 对象的父对象
    ProgressBar1.Parent := StatusBar1;
    //设置 ProgressBar1 对象相关的位置属性
    ProgressBar1.Top := 2;
```

```
    ProgressBar1.Height := StatusBar1.Height - 2;
    //显示于 StatusBar1.Panels[0]
    ProgressBar1.Left := 0;
    ProgressBar1.Width := StatusBar1.Panels[0].Width;
    //设置 ProgressBar1 对象的范围及重复的次数
    ProgressBar1.Min := 0;
    ProgressBar1.Max := Total;
    ProgressBar1.Step := StepQ; //重复的次数
    //显示 ProgressBar1 对象
    ProgressBar1.Visible := True;
    //显示进度
    for i := 1 to Total  do ProgressBar1.Stepit;
  finally
    ShowMessage('运行完毕!!');
    ProgressBar1.Visible := False;
  end;
end;
```

5 保存并按【F9】键运行工程。工程运行后，当您单击“确定”按钮，程序会将进度显示于 TStatusBar 对象中的 TProgressBar 对象，如图 15.6 所示。

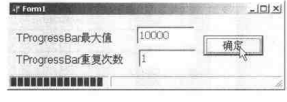


图 15.6　TProgressBar 对象与 TStatusBar 对象结合范例



上面的范例示范如何将 TProgressBar 对象显示于 TStatusBar 对象的第一个 Panel 中。如果您想将它放置于其他的 Panel 中，请参考下列的代码：

```
//显示于 StatusBar1.Panels[1]
//ProgressBar1.Left := StatusBar1.Panels[0].Width + 2;
//ProgressBar1.Width := StatusBar1.Panels[1].Width;
```

15.4　TTimer 对象

在本小节中主要是介绍 TTimer 对象，此对象位于 System 的组件面板上，TTimer 对象每隔一段时间会自动去调用一个事件，而间隔时间是由用户自定义的。我们常利用此项特性来使程序定期去处理某项事件，例如我们可以作出类似于码表的功能，让 TTimer 对象自动去计算时间。以下就 TTimer 对象经常被使用的属性加以说明。

Enabled 属性

设置 TTimer 对象是否有效，属性值为 True 时为有效，在一定时间内自动调用 OnTimer

事件。若属性值为 False 时便不会调用 OnTimer 事件。

Interval 属性

Interval 属性是设置 OnTimer 事件触发时的间隔时间，其单位为千分之一秒。因为 Interval 属性是一个 Cardinal 的整数类型，因此其范围从 0~4294967295。当 Interval 属性为非 0 时才会去调用 OnTimer 事件，属性默认值为 1000，即每秒会触发一次 OnTimer 事件。

范例 15-5

目的：本范例通过定时器的制作来说明 TTimer 对象各项属性的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入两个 TButton 对象、两个 TLable 对象及一个 TTimer 对象，其属性值设置如表 15.4 所示(对象摆放位置请参见图 15.7)。

表 15.4

对象	属性名称	属性设置值
TForm	Caption	定时器
TButton	Name	PlayStopbutton
	Caption	开始计时
TButton	Name	Resetbutton
	Caption	重算计时
TLable	Name	Lable
	Cpation	累计时间为
TLable	Name	Showtime
	Caption	00:00:00
	Font.Size	24
TTimer	Name	Timer1
	Enable	False

STEP 3 在 var 之后声明六个变量，分别用来保存累计的时间与显示的时间：

```
var
  Form1: TForm1;
  s, m, h: integer;     //累计时间变量
  st, mt, ht : string;  //显示时间变量
```

STEP 4 在 PlayStopButton 的 OnClick 事件处理过程中加入以下代码：

```
procedure TForm1.PlayStopButtonClick(Sender: TObject);
begin
  //当按下开始计时按钮时，会启动 TTimer 对象.
  //并将按钮标题改为停止计时
  if PlayStopButton.caption ='开始计时' then
```

```
    begin
      PlayStopButton.Caption := '停止计时';
      Timer1.Enabled := True;
    end
  else
    begin
      //若按下的是停止按钮时，会停止 TTimer 对象，
      //并将按钮标题改为开始计时
      PlayStopButton.Caption :='开始计时';
      Timer1.Enabled := False;
    end;
end;
```

步骤 5 在 ResetButton 对象的 OnClick 事件处理过程中加入以下代码：

```
procedure TForm1.ResetButtonClick(Sender: TObject);
begin
//当按下重新计时按钮时，会停止 TTimer 对象，并设回默认值
  Timer1.Enabled := False;
  ShowTime.Caption := '00:00:00';
  s:=0;
  m:=0;
  h:=0;
  PlayStopButton.Caption := '开始计时';
end;
```

步骤 6 在 Timer1 的 OnTimer 事件处理过程中加入以下的代码：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  //当 TTimer 对象启动时，会自动增加一秒
  s := s + 1;

  //当累计为 60 秒时，会自动进位
  if s=60 then
    begin
      m := m + 1;
      s := 0;
    end;

  // 当累计为 60 分时，会自动进位
  if m=60 then
    begin
      h := h + 1;
      m := 0;
    end;
```

```
    //当秒数不为10秒时，在秒数前自动填入0
    if (s div 10)=0 then
      St := '0' + IntToStr(s)
    else
      st := IntToStr(s);

    //当分时不为10分时，在分数前自动填入0
    if (m div 10) = 0 then
      mt := '0' + IntToStr(m)
    else
      mt := IntToStr(m);

    //当时不为10时，在时前自动填入0
    if (h div 10) = 0 then
      ht := '0' + IntToStr(h)
    else
      ht := IntToStr(h);

    //显示累计时间
    ShowTime.Caption := ht + ':' + mt + ':' + st;
  end;
```

(7) 在 Form1 的 FormCreate 事件处理过程中填入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //变量初值设置
  s:= 0;
  m:= 0;
  h:= 0;
end;
```

(8) 保存工程，并按【F9】功能键运行工程。工程运行后，当您单击“开始计时”按钮时，会每隔一秒触发 TTimer 对象的 OnTimer 事件，并将累计时间显示在 TLable 上。若单击“停止计时”按钮时，会停止调用 OnTimer 事件并停止计时。若单击“重新计时”按钮时，会重新调用 OnTimer 事件并且重新设置所有的变量，如图 15.7 所示。



图 15.7 利用 TTimer 对象制作定时器

15.5　TSplitter 对象

TSplitter 对象允许用户在运行时期调整系统所提供的接口，我们通常使用 TSplitter 对象的 Align 属性来决定其摆放的位置。如果用户在运行时期移动 TSplitter 对象，则会触发该对象的 onMove 事件。另外 ResizeStyle 属性可决定对象被移动时鼠标的表现方式，分述如下（见图 15.8）：

- ResizeStyle 属性值为 rsNone 时，表示对象被移动时并没有其他的效果，只显示鼠标光标而已。
- ResizeStyle 属性值为 rsLine 时，表示对象被移动时会出现一条实线及显示鼠标光标。
- ResizeStyle 属性值为 rsPattern 时，表示对象被移动时会出现一条虚线及显示鼠标光标。
- ResizeStyle 属性值为 rsUpdate 时，表示对象会被直接改变大小。

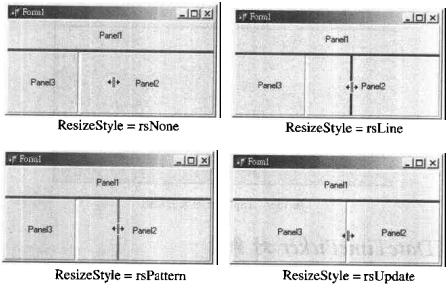


图 15.8　TSplitter 对象的 ResizeStyle 属性范例

以下通过范例来说明如何运用 TSplitter 对象提高应用系统的亲和性。

范例 15-6

目的：本范例主要说明 TSplitter 对象的使用方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象的上方加入一个 TPanel 对象，将其 Name 属性设为 Panel1，并将其 Align 属性设为 alTop。

步骤 3 接着，在 TForm 对象的空白处加入一个 TSplitter 对象，并将其 Align 属性设为 alTop，使其紧靠着 Panel1 的下方，为了便于分辨，我们将 TSplitter 对象的 Color 属性设为

clRed。

4. 在 TForm 对象的空白处加入一个 TPanel 对象，将其 Name 属性设为 Panel2，并将其 Align 属性设为 alRight。

5. 接着，在 TForm 对象的空白处加入一个 TSplitter 对象，并将其 Align 属性设为 alRight，使其紧靠着 Panel2 的左方，为了便于分辨，我们将 TSplitter 对象的 Color 属性设为 clYellow。

6. 最后，在 TForm 对象的空白处加入一个 TPanel 对象，将其 Name 属性设为 Panel3，并将其 Align 属性设为 alClient。

7. 保存工程。

8. 按【F9】功能键运行工程。工程运行后，您可以用鼠标按住 TSplitter 对象以调整对象的摆设位置，如图 15.9 所示。

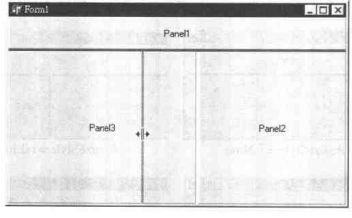


图 15.9 TSplitter 对象的范例

15.6 TDateTimePicker 对象

在应用程序开发中，常常需要用户输入日期或时间的数据，Delphi 6 提供了 TDateTimePicker 对象、TMonthCalendar 对象供您使用。本小节先介绍 TDateTimePicker 对象，而 TMonthCalendar 对象待下一小节再详细说明。TDateTimePicker 对象位于组件面板上的“Win32”标签页中，以下就 TDateTimePicker 对象常用的属性与事件加以说明。

CalAignment 属性

CalAignment 属性代表“日历”出现的位置，其包含 dtaLeft 及 dtaRight 等。前者代表“日历”出现的位置为 TDateTimePicker 对象的左边，而后者代表右边，如图 15.10 所示。

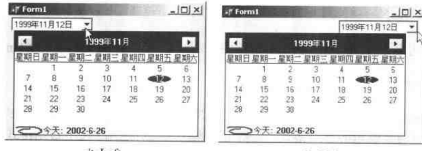


图 15.10　TDateTimePicker 对象的 CalAignment 属性值

CalColors 属性

CalColors 属性继承自 TMonthCalColors 对象，其包含 BackColor、MonthBackColor、TextColor、TitleBackColor、TitleTextColor 及 TrailingTextColor 等。BackColor 代表背景颜色、MonthBackColor 代表日历的背景颜色、TextColor 代表“数字”与“今天日期”的文字颜色、TitleBackColor 代表“标题”的背景颜色、TitleTextColo 代表“标题”的文字颜色、TrailingTextColor 代表非本月份的数字颜色，如图 15.11 所示。



图 15.11　TDateTimePicker 对象的 CalColors 属性

Kind 属性

Kind 属性可决定 TDateTimePicker 对象为日期或时间的类型，其最大的差别在于前者有“日历”可供选择，而后者没有，如图 15.12 所示。

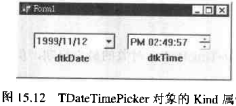


图 15.12　TDateTimePicker 对象的 Kind 属性

Date 属性

Date 属性可取得 TDateTimePicker 对象的日期，但仅在 Kind 属性为 dtkDate 时有效。

- *Time 属性*

Date 属性可获取 TDateTimePicker 对象的时间，但仅在 Kind 属性为 dtkTime 时有效。

- *DateFormat 属性*

DateFormat 属性可决定以长(dfLong)或短(dfShort)的日期格式来获取 TDateTimePicker 对象的日期。这里的长或短的日期格式，主要是受操作系统所控制，关于日期格式，请查看“控制面板”中的“区域选项”，如图 15.13 所示。



图 15.13　控制面板的区域设置会影响 DateFormat 属性

- *DateMode 属性*

DateMode 属性可决定当 TDateTimePicker 对象在 Kind 属性为 dtkDate 时的显示方式，其包含 dmComboBox 与 dmUpDown 两种。前者代表以 ComboBox 方式显示，而后者则以 UpDown 方式显示。

- *MaxDate 属性*

MaxDate 属性可设置 TDateTimePicker 对象的最大日期，但仅在 Kind 属性为 dtkDate 时有效。

- *MinDate 属性*

MinDate 属性可设置 TDateTimePicker 对象的最小日期，但仅在 Kind 属性为 dtkDate 时有效。

- *onChange 事件*

当 TDateTimePicker 对象内容值被更改时，onChange 事件即被触发。

范例 15-7

目的：本范例主要说明 TDateTimePicker 对象的使用方法。

步骤 1 创建新的工程。

步骤 2 在工程中加入一个 TDateTimePicker 对象及 TMemo 对象，其相关属性设置如表 15.5 所示。

表 15.5

对象名称	属性名称	属性设置值
TDateTimePicker	Name	DateTimePicker1
TMemo	Name	Memo1
	Lines	(空白)

步骤 3 在 TDateTimePicker 对象的 onChange 事件处理过程中加入以下的代码：

```
procedure TForm1.DateTimePicker1Change(Sender: TObject);
begin
  //记录时间
  Memo1.lines.add(DateToStr(DateTimePicker1.Date));
end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程。在您选择日期的同时，onChange 事件会不断被触发，如图 15.14 所示。



图 15.14　TDateTimePicker 对象的 onChange 事件

15.7　TMonthCalendar 对象

TMonthCalendar 对象与 TDateTimePicker 对象在功能上相当类似，用法上也差不多，同样是为了方便用户选取日期数据。惟一的不同在于 TMonthCalendar 对象只能选取日期，而不能选取时间。以下是 TMonthCalendar 对象各种常用的属性、操作方法与事件：

AutoSize 属性

AutoSize 属性决定 TMonthCalendar 对象是否有 AutoSize 的能力。

CalColors 属性

CalColors 属性继承自 TMonthCalColors 对象，其包含 BackColor、MonthBackColor、TextColor、TitleBackColor、TitleTextColor 及 TrailingTextColor 等。详细说明，请参考第 15.6 节 TDateTimePicker 对象的 CalColors 属性。

MaxDate 属性

MaxDate 属性决定 TMonthCalendar 对象所允许的最大日期。

MinDate 属性

MinDate 属性决定 TMonthCalendar 对象所允许的最小日期。

FirstDayofWeek 属性

FirstDayofWeek 属性代表 TMonthCalendar 对象以星期几作为一周的第一天。其包含 dowMonday(星期一)、dowTuesday(星期二)、dowWednesday(星期三)、dowThursday(星期四)、dowFriday(星期五)、dowSaturday(星期六)、dowSunday(星期日)及 dowLocaleDefault(操作系统默认值)。

ShowToday 属性

当 ShowToday 属性为 True，表示 TMonthCalendar 对象会显示今天的日期；反之，则不显示。

ShowTodayCircle 属性

当 ShowTodayCircle 属性为 True，表示 TMonthCalendar 对象以红色圈圈标示今天的日期；反之，则不标示。

WeekNumbers 属性

当 WeekNumber 属性为 True，表示 TMonthCalendar 对象标示每周的次序号码；反之，则不标示。

MultiSelect 属性

当 MultiSelect 属性为 True，表示 TMonthCalendar 对象允许用户配合 Shift 功能键选取某一段的日期；反之，则只能选取一个日期。

EndDate 属性

EndDate 属性通常配合 MultiSelect 属性为 True 时使用，EndDate 属性可以获取 TMonthCalendar 对象结束日期。

BoldDays 操作方法

BoldDays 操作方法可用来设置 TMonthCalendar 对象每个月特别指定的数个重要日期。

通常配合 OnGetMonthInfo 事件一起使用。

☆ *onGetMonthInfo 事件*

onGetMonthInfo 事件通常在 TMonthCalendar 对象显示时被触发。

范例 15-8

目的：本范例主要说明 TMonthCalendar 对象的使用方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TMonthCalendar 对象、两个 TLabel 对象及两个 TEdit 对象，其相关属性设置如表 15.6 所示。

表 15.6

对象名称	属性名称	属性设置值
TMonthCalendar	Name	MonthCalendar1
	MultiSelect	True
TLabel	Name	Label1
	Caption	开始日期
TLabel	Name	Label2
	Caption	结束日期
TEdit	Name	EditStart
TEdit	Name	EditEdit

STEP 3 在 TMonthCalendar 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.MonthCalendar1Click(Sender: TObject);
begin
   //选取某一段日期
   EditStart.text := DateToStr(MonthCalendar1.Date);
   EditEnd.text := DateToStr(MonthCalendar1.EndDate);
end;
```

STEP 4 在 TMonthCalendar 对象的 GetMonthInfo 事件处理过程中加入以下的代码：

```
procedure TForm1.MonthCalendar1GetMonthInfo(Sender: TObject; Month: Cardinal;
var MonthBoldInfo: Cardinal);
begin
  //设置重要的日期
  MonthCalendar1.BoldDays([1,15], MonthBoldInfo);
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程，工程运行后，您会发现在每个月的 1 日和 15 日的字体

都比较粗，那是因为我们使用 BoldDays 操作方法的缘故。除此之外，当您选取某一段日期时，系统会在 TEdit 对象中显示开始与结束的日期，如图 15.15 所示。



图 15.15　TMonthCalendar 对象 MultiSelect 属性

范例 15-9

目的：本范例主要说明如何不受操作系统“区域选项”的影响，正确地获取 TMonthCalendar 对象的日期。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TMonthCalendar 对象、两个 TButton 对象、一个 TLabel 对象及一个 TEdit 对象，其相关属性设置如表 15.7 所示。

表 15.7

对象名称	属性名称	属性设置值
TMonthCalendar	Name	MouthCalendar1
TButton	Name	ButtonSet
	Caption	设置日期
TButton	Name	ButtonGet
	Caption	获取日期
TLabel	Name	Label1
	Caption	日期格式为 YYYY/MM/DD
TEdit	Name	EditDate
	Text	(空白)

(3) 在 ButtonSet 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonSetClick(Sender: TObject);
begin
  try
    //设置用户所指定的日期
    MonthCalendar1.Date :=EncodeDate(
            strtoint(copy(EditDate.text,1,4)),
            strtoint(copy(EditDate.text,6,2)),
            strtoint(copy(EditDate.text,9,2)));
```

```
    except
      //当用户所设置的格式不符，则显示今天的日期
      MonthCalendar1.Date := date;
    end;
  end;
```

步骤 4　在 ButtonGet 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonGetClick(Sender: TObject);
var
  Year  : word;
  month : word;
  day   : word;
  MM  : string;
  DD  : string;
begin
  //获取 MonthCalendar1 的日期
  decodedate(MonthCalendar1.Date,year,month,day);

  //重新设置“月份”的显示格式
  if month < 10  then
    MM := '0' + format('%1S',[inttostr(month)])
  else
    MM := format('%2S',[inttostr(month)]);

  //重新设置“日期”的显示格式
  if day < 10  then
    DD := '0' + format('%1S',[inttostr(day)])
  else
    DD := format('%2S',[inttostr(day)]);

  //显示新的日期数据
  EditDate.text := format('%-4S',[inttostr(year)])+'/'+MM+'/'+DD;
end;
```

步骤 5　保存工程。

步骤 6　按【F9】功能键运行工程。在工程运行后，您可以设置 TMonthCalendar 对象的日期，也可以获取 TMonthCalendar 对象的日期，如图 15.16 所示。



图 15.16　TMonthCalendar 对象应用范例

15.8 TFrame 对象

TFrame 对象是一个容器对象(container)，您可以在 TFrame 对象中加入各种组件以作为自定义的模板，这样的样板可以被重复使用于整个工程，也可以重复使用于同一个 TForm 对象中。以下将以实例说明如何使用 TFrame 对象。

范例 15-10

目的：本范例主要说明如何使用 TFrame 对象。

步骤 1 创建新的工程。

步骤 2 从“File/New/Frame”下拉菜单中添加一个 TFrame 对象，其默认名称为 Frame2。

步骤 3 在 Frame2 对象中加入一个 TGroupBox 对象、三个 TLabel 对象、三个 TEdit 对象及两个 TButton 对象，其相关属性设置如表 15.8 所示，相关对象的摆放位置如图 15.17 所示。

表 15.8

对象名称	属性名称	属性设置值
TGroupBox	Name	GroupBox
TLabel	Name	Label1
	Caption	学号
TLabel	Name	Label2
	Caption	姓名
TLabel	Name	Label3
	Caption	科系
TEdit	Name	Edit1
TEdit	Name	Edit2
TEdit	Name	Edit3
TButton	Name	Button1
	Caption	确定
TButton	Name	Button2
	Caption	取消



图 15.17 利用 TFrame 对象创建自定义样板

Step 4 为 Frame2 对象中 Button2 对象的 onClick 事件处理过程加入以下的代码，这段代码主要的目的是清除 Edit1 对象的内容。这里将会与后续的步骤进行比较，详细说明请参考后续的说明。

```
procedure TFrame2.Button2Click(Sender: TObject);
begin
    Edit1.text := '';
end;
```

Step 5 请在 TForm 对象中加入一个 Frames 组件，其位于组件面板的“Standard”页标签中。此时会出现一个“Select frame to insert”对话框，请选择 Frame2 并单击“OK”按钮，如图 15.18 所示。



图 15.18　Select frame to insert 对话框

Step 6 重复步骤 5，再为 TForm 对象加入一个 Frames 组件，如图 15.19 所示。

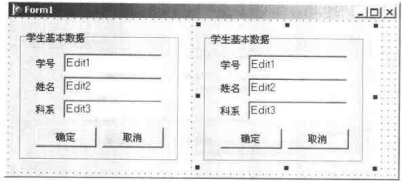
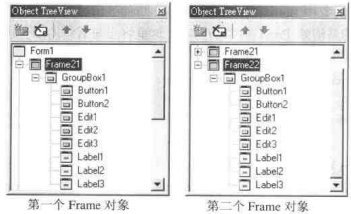


图 15.19　TForm 对象可同时包含多个 TFrame 对象

Step 7 接着，我们将比较一下这两个 Frame 对象与其所附属的对象有何不同。从“Object Tree View”中，您可以发现这两个 TFrame 对象默认的名称分别为 Frame21 与 Frame22，如果您不喜欢这样的命名方式可自行修改。另一个比较的重点是这两个 TFrame 对象所附属的对象，这些附属对象将会保留原始的名称，这样的重复命名方式并不会发生访问上的冲突。同样的，您也可以针对每一个对象重新命名，如图 15.20 所示。



第一个 Frame 对象　　第二个 Frame 对象

图 15.20　比较两个 TFrame 对象的附属对象

步骤 8 接下来，我们将比较这两个 Button2 对象的 onClick 事件处理过程内容不同之处。在图 15.21 中您可以发现系统分别为这两个 Button2 对象的 onClick 事件处理程序加入了类似的代码。这些代码主要是调用派生类中的 Button2Click 事件处理过程。什么是派生类呢?所谓派生类是一个继承自其他类的对象，例如:Frame21 与 Frame22 皆为 Unit2 中 Frame2 的派生类，因为它们都继承自 Frame2 对象。工程运行后，您可以到调试模式中检查过程调用的顺序，事实上，最终都是调用 Unit2 中 Frame2 的 onClick 事件处理过程。通过这样的说明，相信您一定会发现，每一个 TFrame 可以拥有属于自己的代码，当其被继承时同样适用，而无需修改代码，如图 15.21 所示。

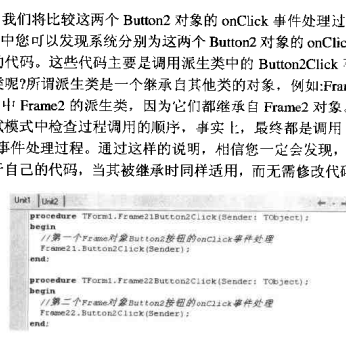


图 15.21　比较两个 Button2 的 onClick 事件处理过程

步骤 9 如果您想直接修改 Frame 中附属对象的属性值，请参考下面程序的示范。请在 TForm 对象中加入一个 TButton 对象，并在其 onClick 事件处理中加入下面的代码。这段程序主要是说明如何同时修改 Frame21 与 Frame22 中 Edit1 对象的 Text 属性值。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    //清除第一个 Frame 的学号字段
    Frame21.Edit1.Text := '';
    //清除第二个 Frame 的学号字段
    Frame22.Edit1.Text := '';
end;
```

步骤 10 保存并按【F9】键运行工程。工程运行后，您可分别运行“取消”按钮清除学号字

段，也可以按下 Form1 的 Button1 对象同时清除两个学号字段。

15.9　TApplication 对象

Application 是 TApplication 类的一个实例(Instance)，而 TApplication 类则被定义于 Forms 程序单元文件中。当 Delphi 工程被运行时，系统会自动创建一个 Application 对象，您可以通过此对象来获取与工程相关的信息。以下就 TApplication 对象常用的属性、操作方法或事件加以说明。

ExeName 属性

ExeName 属性可获取此应用程序运行时的完整路径，无论是直接运行.EXE 或是通过快捷方式运行应用程序。在实际操作中，我们会通过 ExtractFilePath 函数截取该路径的所在目录以利于调用其他的文件，例如:Ini 文件或 Access 数据库。

Icon 属性

Icon 属性可动态改变此应用程序的显示图标。

Title 属性

Title 属性可动态改变此应用程序的显示标题。

范例 15-11

目的：本范例主要说明如何使用 TApplication 对象的 ExeName、Icon 及 Title 等属性。

1. 创建新的工程。
2. 在 TForm 对象上加入一个 TLabel、一个 TEdit 及一个 TButton 对象，其相关属性设置如表 15.9 所示。

表 15.9

对象名称	属性名称	属性设置值
TLabel	Name	Label1
	Caption	完整路径
TEdit	Name	Edit1
TButton	Name	Button1

3. 在 Button1 对象的 onClick 事件处理过程中加入下面的代码。这段程序主要是获取应用程序的运行路径，并从该路径中加载 CONSTRUC.ICO 图标及更改应用程序的标题文字。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  DirPath : String;
```

```
begin
  //显示运行文件位置
  Edit1.Text := Application.ExeName;
  //获取目录名称
  DirPath := ExtractFilePath(Application.ExeName);
  //加载图示
  Application.Icon.LoadFromFile(DirPath + 'CONSTRUC.ICO');
  //更改应用程序标题文字(注意任务栏的变化)
  Application.Title := '新的标题文字';
end;
```

4 保存并按【F9】键运行工程。工程运行后，请比较按下 Button1 按钮前后该应用程序的变化，如图 15.22 所示。

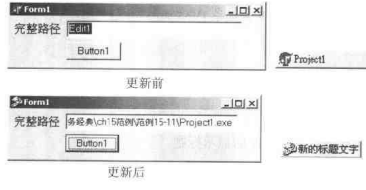


图 15.22 比较应用程序的图标及标题文字

MessageBox 操作方法

MessageBox 操作方法可显示一个属性 Application 的对话框，有关 MessageBox 操作方法请参考第 10.3 节的详细说明。

HandleMessage 操作方法

当操作系统的信息进入信息队列(message queue)时， HandleMessage 操作方法可打断应用程序的运行以利于处理系统的信息。

ProcessMessages 操作方法

ProcessMessages 操作方法可打断应用程序的运行以便处理系统的信息队列(message queue)。

范例 15-12

目的：本范例主要是说明如何允许应用程序被打断，以便处理系统的信息。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TEdit 对象及一个 TButton 对象。
3. 在 TButton 对象的 onClick 事件处理过程中加入以下的代码。这段程序是一个无穷循环，假如没有在循环中加入 ProcessMessages 或 HandleMessage 等操作方法时，您可能无法查看 TEdit 对象的内容值，而且程序几乎是当掉的感觉。但是，假如您加上 ProcessMessages 或 HandleMessage 等操作方法时，您便可以清楚地查看 TEdit 对象的内容值，如下所示（见图 15.23）。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
begin
  //这是一个无穷循环
  while True do begin
    if i = 1000 then i := 1;
    i := i + 1;
    //两者的功能相同
    Application.HandleMessage;
    //Application.ProcessMessages;
    Edit1.text := Inttostr(i);
  end;
end;
```



图 15.23　允许应用程序被打断的运行范例

- ***Terminate* 操作方法**

Terminate 操作方法终止应用程序的运行。

- ***ShowException* 操作方法**

ShowException 操作方法可显示一个指定的异常。

- ***OnException* 事件**

当应用程序发生异常而且没有被适当处理时 Application 对象的 OnException 事件即被触发。

范例 15-13

目的：本范例主要说明如何捕获应用程序未被处理的异常。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TButton 对象，并在其 onClick 事件处理过程中加入以下代码。这段程序主要是触发一个除法运算的异常，此异常将会在 onClick 事件中被显示与处理，且会重新触发它。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y , Z : Extended;
begin
  //写一段会产生异常的程序
  try
   X := 100;
   Y := 0;
   Z := X / Y;
   Showmessage(FloatToStr(Z));
  except
    //处理异常
   On E : Exception do begin
      //显示异常的信息说明
      Showmessage(E.Message);
      //再度触发异常
      Raise;
    end;
  end;
end;
```

STEP 3 为了让 Application 能捕获尚未被处理的异常，因此我们自定义一个名为 myException 过程，并将其指定给 Application 的 OnException 事件。如此一来，Application 就可捕获到尚未被处理的异常。

```
type
  TForm1 = class(TForm)
{略}
  public
    procedure myException (Sender: TObject; E: Exception);
  end;
{略}
procedure TForm1.FormCreate(Sender: TObject);
begin
  //指定 Application 的 onException 所对应的过程
  Application.OnException := myException;
```

```
end;

procedure TForm1.myException(Sender: TObject; E: Exception);
begin
  //显示异常
  Application.ShowException(E);
  //结束程序
  Application.Terminate;
end;
```

④ 保存并运行工程。工程运行后，当您按下 Button1 按钮时，系统会连续触发两个异常，如图 15.24 所示。第一个图为 Button1 的 onClick 所捕获的异常信息，另一个则为 Application 所捕获的异常信息。

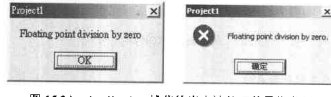


图 15.24　Application 捕获的尚未被处理的异常范例

OnMinimize 事件

当应用程序窗口被最小化时，Application 对象的 OnMinimize 事件即被触发。

Restore 操作方法

Restore 操作方法重新显示应用程序目前的 TForm 对象，通常配合 OnMinimize 事件一起使用。

范例 15-14

目的：本范例主要说明如何使用 OnMinimize 事件及 Restore 操作方法。

① 创建新的工程。

② 在 TForm 对象中加入一个 TTimer 对象，并在其 onTimer 事件处理过程中加入以下的代码。这段程序会重新显示应用程序，并且关闭 TTimer 对象。

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  //重新显示应用程序
  Application.Restore;
  //将 TTimer 关闭
  Timer1.Enabled := False;
end;
```

③ 为了能触发 Application 对象的 OnMinimize 事件。因此我们自定义一个名称为

myMinimize 过程，并将其指定给 Application 的 OnMinimize 事件。如此一来，Application 就可拥有其 OnMinimize 事件处理过程。

```
type
  TForm1 = class(TForm)
{略}
  public
    procedure myMinimize(Sender: TObject);
{略}
procedure TForm1.FormCreate(Sender: TObject);
begin
  //设置 Application 的 OnMinimize 事件处理过程
  Application.OnMinimize := myMinimize;
end;

procedure TForm1.myMinimize(Sender: TObject);
begin
  //设置 TTimer 在三秒钟后启动
  Timer1.Interval := 3000;
  //将 TTimer 启动
  Timer1.Enabled := True;
end;
```

4 保存并运行工程。工程运行后，当您最小化应用程序后，约经过三秒钟的时间，应用程序将自动被重新显示。

15.10 TRegistry 对象

TRegistry 对象可直接访问系统注册文件的内容，在实际操作中，您可以将应用程序中一些可变的参数保存在注册文件中。当这些参数被更改时，您并不需要重新修改与编译程序。例如，当您使用非 BDE 的数据库连接方式，例如 ADO 组件编写数据库应用程序时。因为没有 BDE 别名作为中间接口，因此当其所连接的数据库位置改变时，您的应用程序就需要重新修改与编译。如此一来，就完全失去使用 ADO 组件的好处了，因此您可以将这些数据库连接信息写入系统注册文件中，以便解决这个问题。笔者在第 22 章中，已经将这些访问系统注册文件的功能打包成 VCL 组件，您也可以使用这些组件来访问系统注册文件的数据。

范例 15-15

目的：本范例主要说明如何使用 TRegistry 对象访问系统注册文件。

1 创建新的工程。

2 在 TForm 对象中加入一个 TLabel 对象、一个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 15.10 所示。

表 15.10

对象名称	属性名称	属性设置值
TLabel	Name	Label1
	Caption	数据库位置
TEdit	Name	Edit1
TButton	Name	Button1

步骤 3　为 TButton 对象的 onClick 事件处理过程加入以下的代码，这段程序会将指定的数据项写入系统注册文件中。

```
procedure TForm1.Button1Click(Sender: TObject);
const
  KeyPath = 'CarlosStudio\myDBSetting'; //路径名称
  KeyName = 'DataBase';  //数据名称
var
  myRegistry: TRegistry;
begin
  //创建 TRegistry 对象(uses Registry)
  myRegistry := TRegistry.Create();
  //设置根目录
  myRegistry.Rootkey:=HKEY_CURRENT_USER;
  //打开指定的路径
  myRegistry.OpenKey(KeyPath, True);
  //设置指定 Key 的内容值
  myRegistry.WriteString(KeyName,Edit1.text);
  //释放 TRegistry 对象
  myRegistry.Free;
  Showmessage('数据已保存');
end;
```

步骤 4　保存并按【F9】键运行工程。工程运行后，在您输入适当的 IP 位置后，再按下 Button1 按钮时，程序会将您指定的数据写入系统注册文件中。您可以在命令行中运行 Regedit 通过“注册表编辑器”来查看系统注册文件的内容，如图 15.25 所示。



图 15.25　TRegistry 对象保存范例

STEP 5 接着，笔者将示范如何读取系统注册文件的内容。请在 TForm 对象中声明一个名称为 GetDBIP 的函数。并为此函数加入以下的代码，这段程序从系统注册文件中读取指定的数据项。

```
type
  TForm1 = class(TForm)
(略)
  private
    function GetDBIp : String;
(略)
function TForm1.GetDBIp : String;
const
   KeyPath = 'CarlosStudio\myDBSetting'; //路径名称
   KeyName = 'DataBase';  //数据名称
var
   myRegistry: TRegistry; //uses Registry
begin
   myRegistry := TRegistry.Create(); //创建 TRegistry
   myRegistry.RootKey:=HKEY_CURRENT_USER;//设置根目录
   myRegistry.OpenKey(KeyPath, True); //打开指定路径
   Result := myRegistry.ReadString(KeyName); //读取指定数据
   myRegistry.Free;  //释放
end;
```

STEP 6 上一个步骤所创建的 GetDBIP 函数已经可以正确地获取数据库位置。接下来，笔者要示范如何将其套用于 ADO 的数据库连接组件。首先，请为 TForm 对象加入一个 TADOConnection 对象并参考范例 20-3 创建一个连接至 SQL Server。设置后，其 ConnectionString 属性，也就是数据库的连接信息，如下所示。以下的数据库连接信息可能会因为不同的系统设置而有所不同。

```
Provider=SQLOLEDB.1;Password="";Persist Security Info=True; User ID=sa;
Initial Catalog=pubs;Data Source=CARLOS
```

STEP 7 上一步骤中的 Data Source 就是代表 TADOConnection 对象所要连接的数据库位置，因此我们只要将其替换为 GetDBIP 函数所获取内容即可。请为 TForm 对象再加入一个 TButton 对象，其默认的名称为 Button2，并为其 onClick 事件处理过程加入以下的代码。这段程序将会通过 GetDBIP 函数获取数据库位置并尝试连接至数据库。

```
procedure TForm1.Button2Click(Sender: TObject);
var
  ConnStr : String;
begin
  //组合数据库连接信息
  ConnStr     :=     'Provider=SQLOLEDB.1;Password="";Persist     Security
Info=True;User ID=sa;Initial Catalog=pubs;Data Source=' + GetDBIp;
  ADOConnection1.ConnectionString := ConnStr;
```

```
try
  //打开数据库
  ADOConnection1.Connected := True;
  Showmessage('连接成功');
  //关闭数据库
  ADOConnection1.Connected := False;
except
  Showmessage('无法连接');
end;
end;
```

步骤 8　再一次运行工程。工程运行后，您可以按下 Button2 按钮以获取数据库位置，并尝试连接到指定的数据库。

15.11　TIniFile 对象

TIniFile 对象与 TRegistry 对象同样具有动态保存系统设置值的能力。其惟一的不同，在于前者是写入指定文件名称的 INI 文件，而后者是直接写入系统的注册文件。您可以根据不同的需求来选择适合的方式。一般来说，INI 文件可分为 Section(段)、key(字段名称)及 value(字段值)等三个部分。

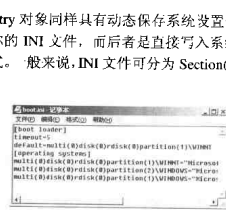


图 15.26　INI 文件范例

从图 15.26 中，您可以发现这个 INI 文件包含 boot loader 和 operation systems 两个区块，在第一个区块中又包括 timeout 与 default 两个字段，其中 timeout 的字段值为 30。如果您以前不曾使用过 INI 文件，相信您现在应该对它有更进一步的认识。以下就 TIniFile 对象常用的属性及操作方法加以说明。

FileName 属性

FileName 是 TIniFile 对象的惟一属性，它记录着 INI 文件保存的完整文件名称与路径。

Create 操作方法

Create 操作方法可动态创建一个 TIniFile 对象，通常在创建的同时会指定其完整文件名称与路径。

ReadSections 操作方法

ReadSections 操作方法可获取指定 INI 文件中所有的区块名称，以图 15.26 而言，

ReadSections 操作方法可获取 boot loader 和 operation systems 两个段名称，其变量保存类型为 TStrings 对象。

- *ReadSection* 操作方法

ReadSection 操作方法可获取指定区块名称的所有字段名称，其变量保存类型为 TStrings 对象。

- *ReadSectionValues* 操作方法

ReadSectionValues 操作方法可获取指定段名称的所有字段名称与字段值，其变量保存类型为 TStrings 对象。

- *EraseSection* 操作方法

EraseSection 操作方法可从特定的 INI 文件中删除指定的整个段内容。

- *SectionExists* 操作方法

SectionExists 操作方法可判断指定的段名称是否存在。

- *ValueExists* 操作方法

ValueExists 操作方法可判断指定的字段值是否存在。

- 数据写入相关操作方法

TIniFile 对象根据不同的数据类型提供不同的写入操作方法。TIniFile 对象所提供的数据写入操作方法包括:WriteBinaryStream、WriteBool、WriteDate、WriteDateTime、WriteFloat、WriteInteger、WriteTime 及 WriteString 等。

- 数据读取相关操作方法

TIniFile 对象根据不同的数据类型提供不同的读取操作方法。TIniFile 对象所提供的数据读取操作方法包括:ReadBinaryStream、ReadBool、ReadDate、ReadDateTime、ReadFloat、ReadInteger、ReadTime 及 ReadString 等。

范例 15-16

目的：本范例主要说明如何使用 TIniFile 对象访问指定的 INI 文件。

(1) 创建新的工程。

(2) 在 TForm 对象上加入三个 TLabel 及三个 TlistBox 对象，其相关属性设置如表 15.11 所示。

表 15.11

对象名称	属性名称	属性设置值
TLabel	Name	Label1
	Caption	所有的区块
TLabel	Name	Label2
	Caption	所有的字段名称
TLabel	Name	Label3
	Caption	所有的字段值
TListBox	Name	ListBox1
TListBox	Name	ListBox2
TListBox	Name	ListBox3

Step 3 在 TForm 对象中声明一个 Public 变量并且在其 onCreate 事件处理过程中加入下列的代码。这段程序主要的目的是打开指定的 INI 文件并且读出其所有的段名称。

```
type
  TForm1 = class(TForm)
(略)
  public
    myIniFile : TINIFile;
(略)
procedure TForm1.FormCreate(Sender: TObject);
var
   IniFileName : String;
begin
  //产生一个与工程相同名称的 Ini 文件
  IniFileName := ExtractFilePath(Application.ExeName) + 'boot.ini';
  //打开一个 ini 文件(uses IniFiles)
  myIniFile := TINIFile.Create(IniFileName);
  //读入所取的 Section
  myIniFile.ReadSections(ListBox1.Items);
end;
```

Step 4 在 TForm 对象 CloseQuery 事件处理过程中加入下列的代码。这段程序主要的目的是释放上一步骤所创建的 TINIFile 对象。

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
   //释放 myIniFile
   myIniFile.Free;
end;
```

Step 5 在 ListBox1 对象的 onClick 事件处理过程中加入下列代码，这段程序主要的目的是获取指定段名称下所有的字段名称与字段值。

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
  FieldName : String;
begin
  if ListBox1.ItemIndex <> -1 then begin
    //获取指定段的所有字段内容
    FieldName := ListBox1.Items[ListBox1.ItemIndex];
    myIniFile.ReadSection(FieldName,Listbox2.Items);
    myIniFile.ReadSectionValues(FieldName,ListBox3.Items);
  end;
end;
```

步骤 6 保存并按【F9】键运行工程。工程运行后，程序会显示指定 INI 文件中的所有段名称。当您选择某个指定段名称时，程序会更进一步地显示其所属的字段内容，如图 15.27 所示。



图 15.27 TINIFile 对象运行范例

范例 15-17

目的：本范例主要说明如何使用 TIniFile 对象保存 TForm 对象的相关属性以便下一次运行时显示。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中声明一个 IniFileName 的字符串变量，其用来保存指定 INI 文件的完整路径名称。

步骤 3 在 TForm 对象的 onCreate 事件处理过程中加入下列的代码，其主要是从指定的 INI 文件中读取 TForm 对象上一次运行时所记录的属性值。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //产生一个与工程相同名称的 Ini 文件
  IniFileName := ChangeFileExt(Application.ExeName,'.INI');
  with TINIFile.Create(IniFileName) do
    try
      //读取上一次 TForm 对象的坐标属性
```

```
      Top := ReadInteger('测试段','Top',Top);
      Left := ReadInteger('测试段','Left',Left);
      Width := ReadInteger('测试段','Width',Width);
      Height := ReadInteger('测试段','Height',Height);
    finally
      //释放 TINIFile 对象
      Free;
    end;
  end;
```

步骤 4　在 TForm 对象的 onCloseQuery 事件处理过程中加入下列的代码，其主要目的是保存目前 TForm 对象的相关属性值。

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  //创建一个 ini 文件(uses IniFiles)
  with TINIFile.Create(IniFileName) do
    try
      //保存目前 TForm 对象的坐标属性
      WriteInteger('测试段','Top',Top);
      WriteInteger('测试段','Left',Left);
      WriteInteger('测试段','Width',Width);
      WriteInteger('测试段','Height',Height);
    finally
      //释放 TINIFile 对象
      Free;
    end;
end;
```

步骤 5　保存并运行工程。工程运行后，您可以任意改变 TForm 的大小或是移动它。工程结束后，您可以发现系统自动创建一个与工程名称相同的 INI 文件，其内容记录着上一次运行时 TForm 对象的相关属性。当您重新运行工程时，TForm 对象会重现最后一次运行状态，如图 15.28 所示。

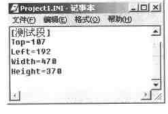


图 15.28　TINIFile 对象记录 TForm 对象的运行属性范例

15.12　TList 对象

在范例 4-5 中我们曾经使用 TList 对象来收集所有新创建(Create)的 TShape 对象，以便能

对 TShape 对象进行操作。本小节便针对 TList 对象作更进一步的介绍。简单地说，TList 对象是一个隐形的对象，为什么说它是隐形的对象呢？因为您无法看到它运行的过程，一切只能靠想象。TList 对象是包含一些可以指到对象的列表(List)，换言之，TList 就像一列火车，在您还没有加入任何对象之前，它只有一个火车头。您所加入的对象就可以当作是一个车厢，至于车厢中您要装什么，没有人会管您的。TList 提供了一些属性(Property)和操作方法(Method)给用户操作，它们包含了增加一个对象到 TList 中、从 TList 中删除一个对象、获取某对象在 List 的位置或是对 TList 中的对象加以排序。表 15.12 和表 15.13 将 TList 对象相关的属性或操作方法加以说明。

表 15.12

属性(Property)	▲ 表示运行时期才能使用	
Capacity	Count	Items
▲List		

表 15.13

操作方法(Method)		
Add	Clear	Delete
Destroy	Error	Exchange
Expand	First	IdexOf
Insert	Last	Move
Pack	Remove	Sort
Create	Free	

以下就 TList 对象常用的属性与操作方法加以说明：

Capacity 属性

用来设置 TList 的容量，如果您没有设置初值，系统会根据内存的大小来自动设置。以我的电脑而言，它会以 4 的个数累加。声明语法如下：

```
property Capacity: Integer;
```

Count 属性

用来计算目前在 TList 中有几个对象，声明语法如下：

```
property Count: Integer;
```

Items 属性

用来指定在 TList 对象中的特定对象，当参数值为 0 代表第一个对象，1 代表第二个对象，依次类推。声明语法如下：

```
property Items[Index: Integer]: Pointer;
```

Add 操作方法

用来添加一个对象到 TList 中，并且返回该对象在 TList 的位置(从 0 开始)。声明语法如下：

```
function Add(Item: Pointer): Integer;
```

Clear 操作方法

删除 TList 中的对象，并且将 Capacity 属性设为 0。声明语法如下：

```
procedure Clear; dynamic;
```

Delete 操作方法

从 TList 中删除某个特定位置(从 0 开始算)的对象，事实上，Delete 并非真正将其删除，它只是把值设为空值(nil)而已。声明语法如下：

```
procedure Delete(Index: Integer);
```

Exchange 操作方法

将存在 TList 中特定的两个对象的位置加以对换。声明语法如下：

```
procedure Exchange(Index1, Index2: Integer);
```

Expand 操作方法

运行时增加 TList 的 Capacity 大小，当 TList 的 Capacity 为 0 时，运行 Exapnd 之后，Capacity 为 4。之后，系统会根据对象个数来决定是否增加 Capacity。当 Count 等于 Capacity 时，expand 才会有效，例如 Count 与 Capacity 同时为 8，则运行 Expand 之后，Capacity 为 16。声明语法如下：

```
function Expand: TList;
```

First 操作方法

返回第一个对象(Returns Items[0])。声明语法如下：

```
function First: Pointer;
```

IndexOf 操作方法

返回某个特定对象存在 TList 的第几个(从 0 起算)，如果不存在则返回-1。声明语法如下：

```
function IndexOf(Item: Pointer): Integer;
```

Insert 操作方法

插入某个特定对象到 TList 特定位置中。声明语法如下：

```
procedure Insert(Index: Integer; Item: Pointer);
```

- *Last* 操作方法

返回 TList 的最后一个对象(Returns Items[Count - 1])。声明语法如下：

```
function Last: Pointer;
```

- *Move* 操作方法

在 TList 中将某个特定对象移到某个特定位置。声明语法如下：

```
procedure Move(CurIndex, NewIndex: Integer);
```

- *Pack* 操作方法

将 TList 中被设为 nil 的项目加以删除，nil 的项目通常是由 Delete 所产生的。

```
procedure Pack;
```

- *Remove* 操作方法

直接用对象指针删除在 TList 中特定的对象，通常用于该对象的 index 值未知的时候。声明语法如下：

```
function Remove(Item: Pointer): Integer;
```

- *Sort* 操作方法

是针对 TList 中的两个特定对象加以比较大小，假如第一个对象小于第二个对象，则返回一个小于零的值，反之，则返回一个大于 0 的值。其声明语法如下：

```
type TListSortCompare = function (Item1, Item2: Pointer): Integer;
procedure Sort(Compare: TListSortCompare);
```

- *Create* 操作方法

创建 TList 对象。声明语法如下：

```
Constructor Create;
```

- *Free* 操作方法

释放 TList 对象。声明语法如下：

```
procedure Free;
```

以下将这些常用的属性与操作方法用一个范例说明。

范例 15-18

目的：本范例示范 TList 对象常用的属性与语法。

(1) 创建新的工程。

(2) 在 Form 对象上加入 13 个 TButton 对象，对象摆放请参考图 15.29，其设置如表 15.14 所示。

表 15.14

对象(Name 属性)	Caption 属性设置值
Tbutton(ButtonCreate)	创建 List(Create)
TButton(ButtonAdd)	加入对象(Add)
TButton(ButtonDelete)	删除最前面的对象(Delete)
TButton(ButtonCount)	查看目前的个数(Count)
TButton(ButtonCapacity)	显示 List 的容量(Capacity)
TButton(ButtonClear)	清除所有对象(Clear)
TButton(ButtonExchange)	将第一个与第二个对象对换(Exchange)
TButton(ButtonExpand)	加大容量(Expand)
TButton(ButtonFirst)	找出第 一个对象(First)
TButton(ButtonLast)	找出最后一个对象(Last)
TButton(ButtonInsert)	插入对象到特定位置(Insert)
TButton(ButtonIndexOf)	测试对象是否存在 MyList(IndexOf)
TButton(ButtonMove)	将最后一个对象移到最前面(Move)

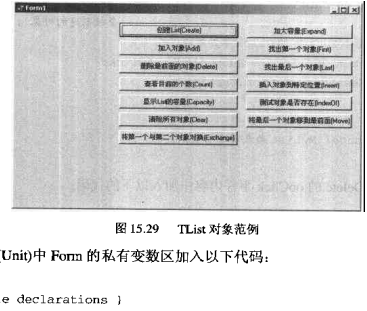


图 15.29　TList 对象范例

(3) 在单元(Unit)中 Form 的私有变数区加入以下代码：

```
private
  { Private declarations }
  MyList:TList;
```

STEP 4 在 ButtonCreate 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonCreateClick(Sender: TObject);
begin
    //创建新的 TList 对象，名叫 MyList
    MyList:= TList.Create;
    ShowMessage('MyList 成功的创建');
    ButtonCreate.enabled:=False;
end;
```

STEP 5 在 ButtonAdd 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonAddClick(Sender: TObject);
var
  MyButton : TButton;
  Temp : Integer;
begin
   //必须先判断 MyList 是否已经创建
   IF MyList <> nil Then
      Begin
        MyButton := TButton.Create(Self);
        MyButton.Parent := Self;
        Temp := MyList.Add(MyButton);
        //设置 MyButton 的各项属性
        MyButton.Top := Temp * 25;
        MyButton.Left  := 0;
        MyButton.Width := 150;
        MyButton.Caption := '第' + IntToStr(Temp) +
                                                '个被创建的对象';
        //设置 MyButton 的 Click 事件处理过程
        MyButton.OnClick := ButtonIndexOfClick;
        ShowMessage(MyButton.Caption);
      End
   Else
      ShowMessage('MyList 尚未创建');
end;
```

STEP 6 在 ButtonDelete 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
var
   Temp : TButton;
begin
   //必须先判断 MyList 是否已经创建
   IF MyList <> nil Then
      Begin
```

```
        //先判断 MyList 所包含的对象个数大于 0
        IF MyList.Count > 0 Then
          begin
            //删除第 0 个对象
            Temp := MyList.Items[0];
            Temp.Free;
            MyList.Delete(0);
          end
        Else
          Showmessage('MyList 中没有对象无法删除');
      End
    Else
      ShowMessage('MyList 尚未创建');
end;
```

7 在 ButtonCount 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonCountClick(Sender: TObject);
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    ShowMessage('MyList 目前有'+IntToStr(MyList.Count)+
                '个对象')
  Else
    ShowMessage('MyList 尚未创建');
end;
```

8 在 ButtonCapacity 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonCapacityClick(Sender: TObject);
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    ShowMessage('MyList 目前的 Capacity 是' +
                IntToStr(MyList.Capacity))
  Else
    ShowMessage('MyList 尚未创建');
end;
```

9 在 ButtonClear 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonClearClick(Sender: TObject);
var
  Temp : TButton;
  I,J  : Integer;
begin
  //必须先判断 MyList 是否已经创建
```

```
    IF MyList <> nil Then
      Begin
        //判断 MyList 的个数是否大于 0
        J := MyList.Count;
        IF J > 0 Then
          Begin
            //先删除所有的对象
            For I:= 0 To MyList.Count -1 do
              Begin
                Temp := MyList.Items[I];
                Temp.Free;
              End;
            //再清除 MyList
            MyList.Clear;
            //显示清除对象的个数
            ShowMessage('已清除了' + IntToStr(J) + '个对象');
          End
        Else
          ShowMessage('目前 MyList 没有包含任何对象、
                                    不需要运行清除(Clear)操作');
      End
    Else
      ShowMessage('MyList 尚未创建');
end;
```

例10 在 ButtonExchange 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonExchangeClick(Sender: TObject);
var
  Temp0,Temp1 : TButton;
  Top0 : Integer;
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    Begin
      //因为要将第 0 个与第 1 个对象交换，
      //故至少要有两个对象才能进行交换
      IF MyList.Count > 1 Then
        Begin
          //记录第 0 个对象的 Top 坐标
          Temp0 := MyList.Items[0];
          Top0 := Temp0.Top;

          //设置第 0 个对象的新坐标与 Caption
```

```
        Temp1 := MyList.Items[1];
        Temp0.Top := Temp1.Top;

        //设置第 1 个对象的新坐标
        Temp1.Top := Top0;

        //实际将 MyList 中的第 0 个与第 1 个对象交换
        MyList.ExChange(0,1);
      End
    Else
      ShowMessage('MyList 中的对象个数太少无法进行交换操作');
  End
Else
  ShowMessage('MyList 尚未创建');
end;
```

(11) 在 ButtonExpand 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonExpandClick(Sender: TObject);
var
  v1 : Integer;
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    begin
      //记录 Expand 之前 MyList 的 Capacity
      v1 := MyList.Capacity;
      MyList.Expand;
      showMessage('MyList 的 Capacity 已经从' + IntToStr(V1) +
                  '扩展到' + IntToStr(MyList.Capacity));
    end
  Else
    ShowMessage('MyList 尚未创建');
end;
```

(12) 在 ButtonFirst 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonFirstClick(Sender: TObject);
var
  Temp : TButton;
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    Begin
      IF MyList.Count > 0 Then
        begin
```

```
            Temp := MyList.First;
            ShowMessage('第 0 个对象的 Caption 是 "' +
                                    Temp.Caption + '"');
          End
        Else
          ShowMessage('MyList 中没有任何对象');
      End
    Else
      ShowMessage('MyList 尚未创建');
end;
```

(13) 在 ButtonLast 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonLastClick(Sender: TObject);
var
  Temp : TButton;
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    Begin
      IF MyList.Count > 0 Then
        begin
          Temp := MyList.Last;
          ShowMessage('最后一个对象的 Caption 是"' +
                                    Temp.Caption + '"');
        End
      Else
        ShowMessage('MyList 中没有任何对象');
    End
  Else
    ShowMessage('MyList 尚未创建');
end;
```

(14) 在 ButtonInsert 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonInsertClick(Sender: TObject);
var
  MyButton : TButton;
  Flag,I :Integer;
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    Begin
      //本范例示范如何插入一个对象到 MyList 中，
      //因此先获取 "中间点"
      Flag := MyList.Count Div 2;
```

```
        //将中间点以后的对象往下移(Top 属性)
        For I:= Flag To MyList.Count - 1 do
            Begin
              MyButton := MyList.Items[I];
              MyButton.Top := (I+1) * 25;
            End;

        //无论是否需要移动中间点以后的对象,
        //都将新的对象插到第 Flag 的对象
        MyButton := TButton.Create(Self);
        MyList.Insert(Flag,MyButton);
        ShowMessage('插入第'+IntToStr(Flag)+'个位置');

        //设置新对象的属性与事件处理过程
        With MyButton do
            Begin
              Parent := Self;
              Top := Flag * 25;
              Width := 150;
              Caption := '使用 Insert 所加入的新对象';
              onClick := ButtonIndexOfClick;
            End;
      End
    Else
      ShowMessage('MyList 尚未创建');
end;
```

(15) 在 ButtonIndexOf 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonIndexOfClick(Sender: TObject);
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
    ShowMessage('第'+IntToStr(MyList.IndexOf(Sender)) +
                '个对象')
  Else
    ShowMessage('MyList 尚未创建');
end;
```

(16) 在 ButtonMove 的 onClick 事件内容中加入以下的代码：

```
procedure TForm1.ButtonMoveClick(Sender: TObject);
begin
  //必须先判断 MyList 是否已经创建
  IF MyList <> nil Then
```

```
    Begin
      IF MyList.Count > 0 Then
        Begin
          //本范例示范如何将最后一个对象移到第一个，与
          //Exchage 不同的是，这里并没有改变对象的显示位置
          //如果您想改变显示位置，请参考 Exchange 范例说明
          MyList.Move(MyList.Count-1,0);
          showMessage('已经完成MyList 中的移动了，您可以'+
                '在该对象上按左键一下测试该对象'+
                                'MyList 中的排列位置');
        end
      Else
        ShowMessage('MyList 中没有任何对象，无法运行移动
                          (Move)操作');
    End
  Else
    ShowMessage('MyList 尚未创建');
end;
```

(17) 在 Form1 的 onClose 事件内容中加入以下的代码：

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  //释放 MyList
  MyList.Free;
end;
```

(18) 保存工程。

(19) 按【F9】键运行工程。工程运行后，您可针对所有的功能一一加以测试。

15.13 TStrings 与 TStringList 对象

TStrings 对象可用来描述一个字符串串行，而 TStringList 对象是一个继承自 TStrings 对象的一个字符串串行，它拥有维护字符串列表的相关功能。图 15.30 说明 TStrings 与 TStringList 对象间的继承关系。

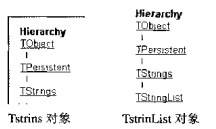


图 15.30 TStrings 对象与 TStringList 对象的继承关系

Delphi 所提供的 VCL 组件中，有很多的属性都与 TStrings 对象有关，例如:TMemo 对象

与 TRichEdit 对象的 Lines 属性、TListBox 对象与 TComboBox 对象的 Items 属性。就如前面所解释的一样，TStrings 对象用来描述一个字符串列表，真正操作字符串列表的则是 TStringList 对象。事实上，TStringList 与 TList 对象在用法上相当的类似，其最大的不同在于前者只能保存字符串，而后者则可以保存对象。以下就 TStringList 对象经常被使用的属性及操作方法加以说明。

Text 属性

Text 属性是指 Lines 属性所有的文字内容。它与 TMemo 及 TRichEdit 等对象的 Text 内容是相同的。

范例 15-19

目的：本范例主要说明如何使用 TStringList 对象的 Text 属性的使用方法。本范例需要两个 TMemo 对象及一个 TButton 对象，并在 TButton 对象的 onClick 事件处理过程中加入以下的代码（见图 15.31）：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //将 Memo1 的内容复制到 Memo2
  Memo2.text:=Memo1.Lines.Text;
end;
```

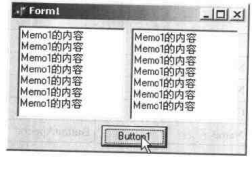


图 15.31　TStrings 对象的 Text 属性

Add 与 Append 操作方法

Add 与 Append 的操作方法同样是将指定字符串加到 TStringList 对象的最后位置。

Clear 操作方法

Clear 操作方法会清除 TStringList 对象中的所有内容。

Delete 操作方法

Delete 操作方法会清除 TStringList 对象中指定行的字符串。

- *Exchange* 操作方法

 Exchange 操作方法会将 TStringList 对象中两个指定行的字符串加以交换。

- *Insert* 操作方法

 Insert 操作方法会在 TStringList 对象中插入指定字符串。

- *LoadFromFile* 操作方法

 LoadFormFile 操作方法允许用户将指定的文件内容加载到 TStringList 对象中。

- *SaveToFile* 操作方法

 SaveToFile 操作方法允许用户将 TStringList 对象内容存入指定的文件。

范例 15-20

目的：本范例主要说明如何使用 TStringList 对象的操作方法，本范例将使用 TMemo 对象来示范 TStringLIst 对象八个常用的操作方法。

Step 1 创建新的工程。

Step 2 在 TForm 对象中加入一个 TMemo 对象及八个 TButton 对象，其属性设置如表 15.15 所示。

表 15.15

对象	属性名称	属性设置值
TMemo	Name	Memo1
TButton	Name	ButtonAdd
	Caption	Add
TButton	Name	ButtonAppend
	Caption	Append
TButton	Name	ButtonClear
	Caption	Clear
TButton	Name	ButtonDelete
	Caption	Delete
TButton	Name	ButtonExchange
	Caption	Exchange
TButton	Name	ButtonInsert
	Caption	Insert
TButton	Name	ButtonLoad
	Caption	Load

续上表

对象	属性名称	属性设置值
TButton	Name	ButtonSave
	Caption	Save
TOpenDialog	Name	OpenDialog1
TSaveDialog	Name	SaveDialog1

STEP 3 在 ButtonAdd 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonAddClick(Sender: TObject);
begin
    //增加字符串到 Memo1 的最后位置
    Mero1.Lines.add('Add String');
end;
```

STEP 4 在 ButtonAppend 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonAppendClick(Sender: TObject);
begin
    //增加字符串到 Memo1 的最后位置
    Memo1.Lines.append('Append String');
end;
```

STEP 5 在 ButtonClear 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonClearClick(Sender: TObject);
begin
    //清除 Memo1 中的所有字符串
    Memo1.Lines.Clear;
    //Memo1.Clear; //两者相同
end;
```

STEP 6 在 ButtonDelete 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
begin
    //删除 Memo1 中的指定字符串(第一行)
    Memo1.Lines.Delete(0);
end;
```

STEP 7 在 ButtonExchange 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonExchangeClick(Sender: TObject);
begin
    //将两个指定行加以交换(第一行与第二行)
    Memo1.Lines.Exchange(0,1);
end
```

STEP 8 在 ButtonInsert 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonInsertClick(Sender: TObject);
begin
   //在指定行之后插入字符串
   Memo1.Lines.Insert(1,'Insert String');
end;
```

(9) 在 ButtonLoad 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonLoadClick(Sender: TObject);
begin
   //打开指定文件
   if OpenDialog1.Execute then
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

(10) 在 ButtonSave 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonSaveClick(Sender: TObject);
begin
   //保存文件
   if SaveDialog1.Execute then
      Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

(11) 保存工程。

(12) 按【F9】功能键运行工程。工程运行之后，您可以对各种操作方法进行测试，包括从文件读取数据及将数据存到文件中，如图 15.32 所示。

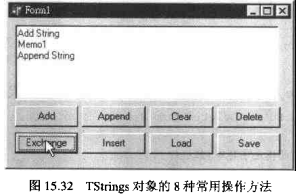


图 15.32 TStrings 对象的 8 种常用操作方法

第16章

MDI 应用程序

本章目标

在本章之前我们所使用的范例被称为 SDI(Single Document Interface)应用程序。一般窗口程序设计除了 SDI 应用程序外，另外有一种经常被使用的应用程序界面我们称它为 MDI(Multiple Document Interface)应用程序。什么是 MDI 应用程序呢?MDI 应用程序可以同时提供多个文件接口供用户使用，最典型的例子就是 Microsoft Word，您可以同时打开多个文件进行编辑。本章首先将为大家介绍如何使用对象库创建 MDI 应用程序，并且以 Step by Step 的方式说明如何手动创建 MDI 应用程序。

本章主题

16.1 多文档界面(MDI)应用程序

16.2 利用对象库建立 MDI 应用程序

16.3 手动创建 MDI 应用程序

16.1 多文档界面(MDI)应用程序

在进入本章之前，我们所使用的范例只能称为单文档界面(SDI，Single Document Interface)，也就是说工程是由单一的 TForm 对象所控制；而您平常所用的文字处理软件(如 Microsoft Word 2000)，用户界面则称为多文档界面(MDI，Multiple Document Interface)。事实上，要开发一个 MDI 应用程序是非常不容易的，但由于 Delphi 6 完整的将 Windows MDI 标准封装在 TForm 对象中，因此可以让初学者轻易的创建 MDI 应用程序。一般来说，一个典型的 MDI 应用程序通常包含二种窗口类型，请参考图 16.1。

应用程序主窗口

应用程序主窗口又称 Frame Window，它是 MDI 应用程序的主要窗口。Frame Window 与一般 SDI 的 TForm 对象同样具有图标、窗体 Title、最小化钮、最大化钮及关闭钮等系统选项。

子窗口管理程序

子窗口管理程序又称为 Client Window，从视觉上来说，Client Window 是子窗口的活动区，也就是说所有的子窗口都只能在这个范围活动，无论是最大化或最小化。从管理上来说，Client Window 负责管理所有子窗口与主窗口之间的沟通。

子窗口

子窗口又称 Child Window，它是 MDI 应用程序真正的工作区，它就像 Word 95/97 中可以打字的区域。它和一般的 SDI 应用程序一样也有图标、窗体 Title、最小化钮、最大化钮及关闭钮等功能外，也有自己的菜单(Menu)。但是，Child Window 的菜单通常会和主窗口的菜单合并在一起，如图 16.1 所示。

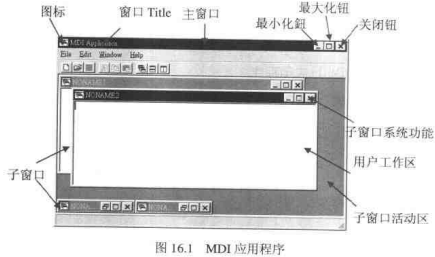


图 16.1 MDI 应用程序

16.2　利用对象库创建 MDI 应用程序

前面曾经提过 Delphi 6 提供一个对象库(Object Repository)让程序设计师可以非常快速的创建 MDI 应用程序，以下先说明制作的过程，再来谈谈其中的细节。

范例 16-1

目的：本范例示范如何使用对象库(Object Repository)创建 MDI 应用程序。

步骤 1 选择菜单栏中的“File/New/Other…”，准备创建新的工程。此时出现“New Items”窗口，请在“Projects”页标签中选择“MDI Application”图标，并单击“OK”按钮，如图 16.2 所示。

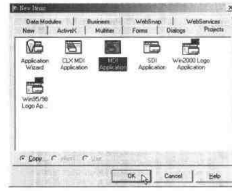


图 16.2　选择 MDI Application 图标

步骤 2 选择此 MDI 工程所要保存目录，并单击“OK”按钮，如图 16.3 所示。



图 16.3　选择所要保存的目录

步骤 3 此时已创建好您的 MDI 工程了，您只要运行工程即可。

在这个 MDI 的主窗口中自动加入了 TMainMenu、TOpenDialog、TStatusBar、TActionList、TAction、ToolBar 及 TImageList 等对象，虽然这些对象大部分您都没看过，但没有关系，因为在以后的章节会陆续说明，如图 16.4 所示。

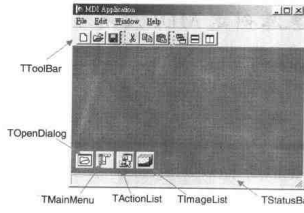


图 16.4 MDI 默认的对象

上面曾经提过，一个 MDI 应用程序包含主窗口、子窗口管理程序及子窗口等。因此您可以通过菜单“View/Project Manager”，打开“Project Manager”窗口，如图 16.5 所示。

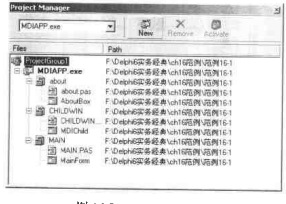


图 16.5 Project Manager

在 Project Manager 窗口中您可以发现 Delphi 6 自动产生了三个 TForm 对象，除了主窗口外，尚有子窗口及 AboutBox 等。子窗口中有一个 TMemo 对象，这主要是用户将来的工作区，也就是输入数据的地方，请参考图 16.6。

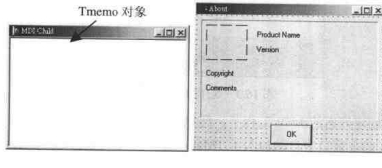


图 16.6 MDIChild 与 AboutBox 对话框

查看 Delphi 6 为 MDI 应用程序所产生的对象后，接着，我们来谈谈 Delphi 6 为 MDI 应用程序所产生的代码。

MDI 主窗口代码

```
unit Main;
interface
//自动加入引用的单元文件
uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,StdCtrls,
Dialogs, Buttons, Messages, ExtCtrls, ComCtrls, StdActns,ActnList, ToolWin,
ImgList;
type
  TmainForm = class(TForm)
    //自动创建一个下拉式菜单及各种选项
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    FileNewItem: TMenuItem;
    FileOpenItem: TmenuItem;
    FileCloseItem: TmenuItem;
    Window1: TMenuItem;
    Help1: TMenuItem;
    N1: TMenuItem;
    FileExitItem: TMenuItem;
    WindowCascadeItem: TMenuItem;
    WindowTileItem: TMenuItem;
    WindowArrangeItem: TMenuItem;
    HelpAboutItem: TMenuItem;
    OpenDialog: TOpenDialog;
    FileSaveItem: TMenuItem;
    FileSaveAsItem: TMenuItem;
    Edit1: TMenuItem;
    CutItem: TMenuItem;
    CopyItem: TMenuItem;
    PasteItem: TMenuItem;
    WindowMinimizeItem: TMenuItem;
    //创建一个状态栏
    StatusBar: TStatusBar;

    //创建一个 TActionList，TActionList 可以提供程序开发者使用及
    //管理内置的 Action，换言之，您可以不用写任何的代码，而应
    //用 Delphi 6 所提供的功能，在本范例中的 File 功能(Close)、Edit
    //功能(Cut,Copy,Paste)及 Window(Cascase,Tile Horizontally
    //Tile Vertically,Minimize All,Arrange All)都是应用现成
    //功能，而不需写程序。
    ActionList1: TActionList;
    //使用 TActionList 中的 16 项 Action
    EditCut1: TEditCut;
```

```
    EditCopy1: TEditCopy;
    EditPaste1: TEditPaste;
    FileNew1: TAction;
    FileSave1: TAction;
    FileExit1: TAction;
    FileOpen1: TAction;
    FileSaveAs1: TAction;
    WindowCascade1: TWindowCascade;
    WindowTileHorizontal1: TWindowTileHorizontal;
    WindowArrangeAll1: TWindowArrange;
    WindowMinimizeAll1: TWindowMinimizeAll;
    HelpAbout1: TAction;
    FileClose1: TWindowClose;
    WindowTileVertical1: TWindowTileVertical;
    WindowTileItem2: TMenuItem;

    //创建一个工具条(ToolBar)，共包含 11 的 ToolButton
    ToolBar2: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
    ToolButton9: TToolButton;
    ToolButton7: TToolButton;
    ToolButton8: TToolButton;
    ToolButton10: TToolButton;
    ToolButton11: TToolButton;

    //创建一个 TmageList,它可以配合 TToolBar,TMainMenu 及
    //TActionList 使用
    ImageList1: TimageList;

//创建四个事件处理过程，包括新建、打开、打开 AboutBox 及
//关闭文件
    procedure FileNew1Execute(Sender: TObject);
    procedure FileOpen1Execute(Sender: TObject);
    procedure HelpAbout1Execute(Sender: TObject);
    procedure FileExit1Execute(Sender: TObject);
  private
    { Private declarations }
    //声明一个私有函数 CreateMDIChild
```

```
    procedure CreateMDIChild(const Name: string);
  public
    { Public declarations }
  end;
var
  MainForm: TMainForm;

implementation
{$R *.DFM}

//引用两个单元，ChildWin 子窗口的单元文件，About 是 AboutBox
//窗口的单元文件
uses ChildWin, About;

//声明一个私有函数 CreateMDIChild，本函数主要的功能是创建
//一个新的子窗口，子窗口的 Caption 属性，由参数 Name 来决定
procedure TMainForm.CreateMDIChild(const Name: string);
var
  //声明一个为 TMDIChild 类型的变量，而 TMDIChild 类型
  //则是子窗口，其定义在 ChildWin 单元文件中
  Child: TMDIChild;
Begin
  //创建一个新的 MDI 子窗口{MDI child window}
  Child := TMDIChild.Create(Application);

  //设置这个子窗口的 Caption 属性
  Child.Caption := Name;

  //如果这个文件存在，则将文件内容加载到子窗口中的 TMemo 对象
  //有关 FileExists 请参考附录 B
  if FileExists(Name) then Child.Memo1.Lines.LoadFromFile(Name);
end;

procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
  //打开一个新的子窗口，其标题名称是由目前子窗口个数来决定
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
End;

Procedure TMainForm.FileOpen1Execute(Sendor: TObject);
Begin
//利用 OpenDialog 对话框，打开文件
//如果用户有指定文件名，则调用 CreateMDIChild 函数，以打开子窗口
```

```
  if OpenDialog.Execute then
    CreateMDIChild(OpenDialog.FileName);
End;

Procedure TMainForm.HelpAboutExecute(Sender: TObject);
Begin
  //Show 出 AboutBox
  AboutBox.ShowModal;
end;

procedure TMainForm.FileExitExecute(Sender: TObject);
begin
  //结束整个 MDI 应用程序
  Close;
end;
end.
```

通过上面的介绍，我想您一定会赞叹 Delphi 6 强大的功能。没错！只要您利用 Delphi 6 所提供的各项对象，相信您一定可以编写出各种强大功能的应用程序，如此一来程序设计才会变为一件快乐的事。在应用程序主窗口中加入了 TImageList、TActionList、TToolBar、TStatusBar、TMainMenu 及 TOpenDialog 等对象在后面的章节会陆续说明。



如果想让您的 ToolBar 变得更漂亮，请将 TToolBar 对象的 Flat 属性设为 True。如图 16.7 所示。有关 TToolBar 对象请参考第 6 章的详细介绍。

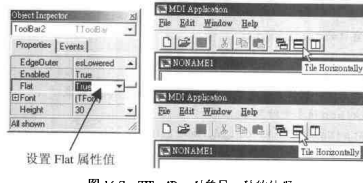


图 16.7 TToolBar 对象另一种的外观

在介绍 MDI 应用程序主窗口的代码之后，接着我们来看看在 MDI 子窗口中究竟需要撰写哪些代码？与查看主窗口代码一样，您可以通过菜单“View/Project Manager”，打开“Project Manager”窗口以方便查看子窗口代码。

子窗口代码

```
//子窗口的单元名称
unit Childwin;
interface
uses Windows, Classes, Graphics, Forms, Controls, StdCtrls;
```

```
type
  //子窗口继承自 TForm 对象，并加入一个 TMemo 对象
  TMDIChild = class(TForm)
    Memo1: TMemo;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
implementation
{$R *.DFM}

procedure TMDIChild.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  //结束子窗口. 这里的结束方法与一般的 TForm 对象有些不同
  Action := caFree;
end;
end.
```

注意　在子窗口中事实上只有一行代码，由于它是一个 ChildForm(因为它的 FormStyle 被设为 fsMDIChild)，所以要结束它并非使用 Close，而要将其参数值 Action 设为 caFree 即可。

16.3　手动创建 MDI 应用程序

通过范例 16-1 的示范，相信您对 MDI 应用程序应该有些概念了。如果您想自己动手做，按部就班的完成 MDI 应用程序的话，那本小节您是不应该错过的。也许您会问，既然对象库那么好用，我为什么还要示范手动创建 MDI 应用程序呢？这有一个很充分的理由，那就是要让大家能够更详细的了解创建 MDI 应用程序的每一个步骤，如果您能很了解它是如何被组合的，那您才可以从中作更多的变化，例如，您可以利用 MDI 应用程序创建一个全功能的 Office。

要完成一个 MDI 应用程序，可以分为下列几个步骤：

- 创建 MDI 应用程序主窗口。
- 创建 MDI 子窗口。
- 文件功能的设置。
- 创建其他的窗口(如 AboutBox 窗口)。

16.3.1　创建 MDI 应用程序主窗口

主窗口(Frame Window，又称框表格)在 MDI 应用程序中占有相当重要的地位，因为它为 MDI 应用程序的文件提供一个工作区域。换言之，如果 MDI 子窗口是图画纸，那么主窗口

就是您的书桌。因此创建主窗口便是创建 MDI 应用程序的第一步。

范例 16-2

目的：本范例示范如何以手动的方式创建 MDI 应用程序。

步骤 1 创建新的工程。

步骤 2 将 Delphi 6 所预设的 TForm 对象做如表 16.1 所示的设置。

表 16.1

属性名称	属性值
Name	MainForm
Caption	MDI 应用程序范例
FormStyle	fsMDIForm

步骤 3 将工程保存，并设 MainForm 的单元名称为 MainUnit。

步骤 4 在 MainForm 加入一个 TStatusBar 对象(位于组件面板 Win32 页标签中)，将其 Name 属性设为 StatusBar。

步骤 5 在 MainForm 加入一个 TImageList(位于组件面板 Win32 页标签中)，将 Name 属性设为 ImageList1，并加入一些图形。至于如何为 TImageList 对象加入图形文件，请参考图 16.8 的示范。这些图形文件最好能配合您程序功能的图案为佳，但这里用 Delphi 6 所提供的小图标(.ico)做示范，可能有些图案会与功能不搭配，您就别太计较喔。关于 TImageList 对象的详细介绍请参考第 13.2 节。

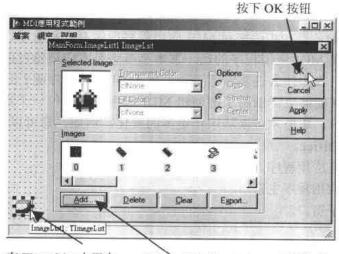


图 16.8 在 TImageList 加入图片

步骤 6 在 MainForm 对象加入一个 TActionList(位于组件面板 Standard 页标签中)，将 Name 属性设为 ActionList1，并设置 ActionList1 的 Action。关于 TActionList 对象请参考

第 14.1 节的详细说明，这里仅示范如何加入窗口排列功能的 Action 为例，详细的步骤请参考图 16.9 所示。

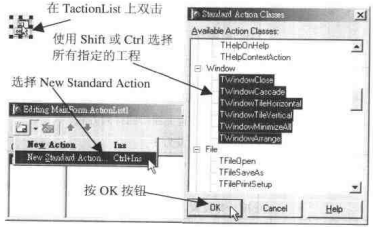


图 16.9　设置 ActionList1 的 Action 选项

(7) 在 MainForm 加入一个 TMainMenu 对象，并加入“文件”、“窗口”及“帮助”等主要选项，有关 TMainMenu 对象的设置如表 16.1 所示。

表 16.1

主功能/次功能	选项设置
文件	Caption 设为“文件”
	Name 设为 FileMenu
	Hint 设为“有关文件的各项功能”
	GroupIndex 设为 0
文件/新建	Caption 设为“新建”
	Name 设为 OpenNewFile
	Hint 设为“新建文件”
文件/打开	Caption 设为“打开”
	Name 设为 OpenFile
	Hint 设为“打开文件”
文件/保存文件	Caption 设为“保存”
	Name 设为 SaveFile
	Hint 设为“保存文件”
文件/另存为	Caption 设为“另存新文件”
	Name 设为 SaveAsFile
	Hint 设为“另存文件”
文件/关闭	Action 设为 Close1
	Caption 设为“关闭”
	Name 设为 CloseFile
	Hint 设为“关闭目前的文件”

续上表

主功能/次功能	选项设置
文件/退出系统	Caption 设为“退出系统” Name 设为 CloseSystem Hint 设为“关闭整个工程”
窗口	Caption 设为“窗口” Name 设为 windows Hint 设为“有关窗口的排列方式” GroupIndex 设为 3
窗口/Cascade	Aciton 设为 Cascade1 (只要您选择 Action 的其中一项，则有关 Name、Caption 等属性会被一并设置。)
窗口/Tile Horizontally	Aciton 设为 TileHorizontally1
窗口/Tile Vertically	Aciton 设为 Tile Vertically1
窗口/Minimize All	Aciton 设为 MinimizeAll1
窗口/Arrage	Aciton 设为 Arrage1
帮助	Caption 设为“帮助” Name 设为 Helps Hint 设为“关于系统的说明” GroupIndex 设为 4
帮助/关于系统	Caption 设为“关于系统” Name 设为 About Hint 设为“关于系统的作者”



图 16.10 MainMenu 设置

注意 有关 Action 的设置步骤这里只示范“窗口/Casecade”的设置，其余的请自行设置，详细说明请参考图 16.11 所示。



1.选择 TmenuItem　　2.选择 Action Item　　3.查看结果

图 16.11　设置 MainMenu 的 Action

注意

为了配合 ChildForm 的菜单选项，因此将 MainForm 菜单选项中“文件”的 GroupIndex 设为 0、“窗口”的 CroupIndex 设为 3、“说明”的 GroupIndex 设为 4。而 ChildForm 的菜单选项“编辑”与“颜色”的 GroupIndex 分别设为 1 与 2。如此一来当 ChildForm 被打开时，它的菜单才会与 MainForm 的菜单结合在一起。

8　如果您希望将来所打开“子窗口名称”可以加到窗口中，例如当您在 Microsoft word 中打开一个文件，系统会自动将文件名称加入下拉式菜单选项。如果您希望您的 MDI 应用程序也有这样的功能，那么您必须使用 MDI 的内建功能，那就是将主窗口的 WindowMenu 属性设为菜单选项中“窗口”的选项即可。本范例的 Name 属性为 windows，如果您找不到 windows 的选项，那么您必须再一次的确认步骤 7 所创建的“窗口”选项的 Name 属性为何?如图 16.12 所示。

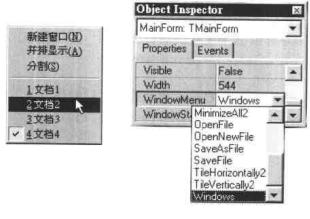


图 16.12　MainForm 的 WindowMenu 设置

9　将 TMainMenu 对象的 Images 属性设为 ImageList1，并在 MainMenu 各功能项中选择适当的图案。在图 16.13 中，以“关于系统”进行示范。

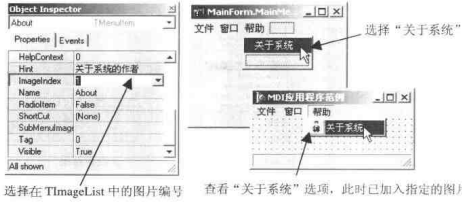


图 16.13 设置 MainMenu 的 Images 选项

STEP 10 如果您希望用户选择菜单时，可以将菜单 Hint 属性的内容 Show 在状态栏的话，那您需要动一下手脚才可以。它的运行原理是这样的：1.先自定义一个过程(Procedure)，2.再将这个过程指定给应用程序的 onHint，其作法如下：

- 将 TStatusBar 对象的 SimplePanel 属性设置为 True。
- 在 TMainForm 对象的私有变量区声明一个 ShowMenuHint 的过程。

```
Private
  { Private declarations }
  procedure ShowMenuHint(Sender: TObject);
```

- 在 ShowMenuHint 过程中加入以下的代码：

```
procedure TMainForm.ShowMenuHint(Sender: TObject);
begin
  //将 MDI 应用程序的 Hint 指定给 StatusBar
  StatusBar.SimpleText:=Application.hint;
end;
```

- 在 TMainForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  //设置 MDI 应用程序 onHint 的函数名称
  Application.onHint := ShowMenuHint;
end;
```

通过上面十个步骤的努力，我们已经创建 MDI 应用程序的主窗口。但这只完成 MDI 应用程序的结构而已。接下来，我们将在下一小节示范如何创建 MDI 应用程序的子窗口。

16.3.2 创建 MDI 子窗口

在 MDI 应用程序主窗口创建后，我们将示范如何创建 MDI 应用程序子窗口。MDI 应用程序子窗口最主要是作为 MDI 应用程序的工作区，虽然用户可以创建很多个子窗口，但我们只要创建一个子窗口模板就可以了。

1 选择菜单“File / New / Form”，为工程加入一个新的 TForm 对象，这个对象上要是作为 MDI 子窗口之用。因此您必须做如表 16.3 所示的设置。

表 16.3

属性名称	属性设置值
Name	ChildForm
FormStyle	fsMDIChild

2 在子窗口(ChildForm)“Project Options”对话框的 Form 页标签，将“ChildForm”设置 Available Form，如图 16.14 所示。

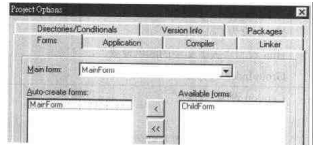


图 16.14　设置 ChildForm 为 Available Form

注意　由于工程运行时，可能会动态产生零个到多个子窗口(ChildForm)，因此我们必须将 ChildForm 设为 Available Form。详细说明请参考第 5 章“5.2.3　TForm 对象产生方式”小节。

3 在 ChildForm 中加入一个 TActionList，并引用与 Edit 有关的 Action，如图 16.15 所示。有关 TactionList 对象的详细设置过程请参考第 16.3.1 节。有关 TImageList 对象在此不再说明，读者可自行加入，使用方法参考第 16.3.2 节，如果您找不到适当的图片可以使用，那么您可以复制范例 16.1 的 TImageList 对象到您的工程中，因为该 TImageList 对象已经包含了一些常用的图片了。

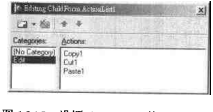


图 16.15　设置 ChildForm 的 ActionList

4 接着设置 ChildForm 的菜单，事实上您也可以将 ChildForm 的菜单放在 MainForm 中，但笔者认为这并不是一个好方法。因为 MDI 子窗口的菜单是针对子窗口而设置的，如果它具有文字处理功能的子窗口，则它必须提供相对的功能，反之如果它具有图像、电子表格、简报等处理功能，那么您就必须提供相对的菜单，所以笔者将其分开设计。所以请在 ChildForm 中加入一个 MainMenu 对象，并加入各项菜单，如表 16.4 所示。请参考图 16.16 所示。

表 16.4

主菜单/子菜单	选项设置
编辑	Caption 设为“编辑”
	Name 设为 EditMenu
	Hint 设为“关于文件的各项编辑功能”
	GroupIndex 设为 1
编辑/剪切	Action 设为 Cut1
编辑/复制	Action 设为 Copy1
编辑/粘贴	Action 设为 Paste1
颜色	Caption 设为“颜色”
	Name 设为 ColorMenu
	Hint 设为“关于文字的颜色设置”
	GroupIndex 设为 2
颜色/红色	Caption 设为“红色”
	Name 设为 red
	Hint 设为“将文字改变红色”
颜色/黑色	Caption 设为“黑色”
	Name 设为 black
	Hint 设为“将文字改变黑色”
颜色/蓝色	Caption 设为“蓝色”
	Name 设为 blue
	Hint 设为“将文字改变蓝色”
颜色/绿色	Caption 设为“绿色”
	Name 设为 green
	Hint 设为“将文字改变绿色”

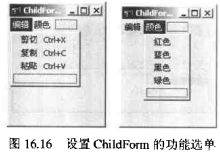


图 16.16　设置 ChildForm 的功能选单



为了配合 MainForm 的菜单，因此将 ChildForm 菜单中“编辑”的 GroupIndex 设为 1、“颜色”的 CroupIndex 设为 2。如此一来当 ChildForm 被打开时，它的菜单才会与 MainForm 的菜单结合在一起。

Step 5 在 ChildForm 中加入一个 TRichEdit 对象(位于组件面板 Win32 页标签中)，将其 Name 属性设为 RichEdit、Align 属性设为 alClient、 Font.Size 设为 12 及 Lines 属性设为

空白。为什么我们不使用 TMemo 对象呢?是因为 TRichEdit 可以提供更多文字编辑的功能，如改变体颜色、大小等，如图 16.17 所示。

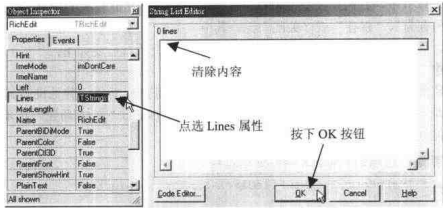


图 16.17　设置 TRichEdit 的 Lines 属性

6 这个步骤所要完成的目的是让用户可以改为在 RichEdit 中的文字颜色，当然 RichEdit 除了文字颜色外，尚可改变其字号、字型名称等。这些细节待以后的章节再一一的说明。请在“红色”、“黑色”、“蓝色”及“绿色”各项功能选项中加入以下的代码：

各种颜色的 onClick 事件处理过程

```
procedure TChildForm.blueClick(Sender: TObject);
begin
  RichEdit.SelAttributes.Color := clBlue;
end;

procedure TChildForm.blackClick(Sender: TObject);
begin
  RichEdit.SelAttributes.Color := clBlack;
end;

procedure TChildForm.greenClick(Sender: TObject);
begin
  RichEdit.SelAttributes.Color := clGreen;
end;
```

7 在子窗口中还有一件非常重要的工作，那就是结束 ChildForm。请在 ChildForm 的 onClose 事件处理过程中加入以下的代码：

```
procedure TChildForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Action:=caFree;
end;
```

16.3.3 主窗口文件功能设置

在 16.3.1 章节中，我们曾经在主窗口中创建几个与文件功能有关的选项，包括“新建”、“打开”、“另存为”、“保存”或“关闭”等。其中“关闭”已经利用 TActionList 默认的关闭文件的功能，因此这个部分就不用再作任何设置了。其余项目分述如下：

1. 新建

 新建的功能较为简单，其实就是创建一个 ChildForm 对象，并且设置其 Caption 属性即可，所以请您在 MainForm 对象中的 MainMenu“文件/新建”选项的 onClick 事件处理过程中加入以下的代码：

“文件/新建”选项的 onClick 事件处理过程

```
procedure TMainForm.OpenNewFileClick(Sender: TObject);
var
  //声明一个为 TChildForm 类型的变量，而 TChildForm 类型
  //则是子窗口，其定义在 ChildUnit 单元文件中
  Child: TChildForm;
begin
  //创建一个新的 MDI 子窗口(MDI child window)
  Child := TChildForm.Create(Application);

  //设置这个子窗口的 Caption 属性
  Child.Caption := '未命名' + IntToStr(MDIChildCount);
end;
```

2. 打开

 打开旧文件的功能稍微复杂了一点，但其实也还好啦！与“新建”惟一的不同就是要让用户选择一个所要打开的文件，并且再将文件的内容载入而已。因此，我们必须在 MainForm 上加入一个 TOpenDialog 对象(位于组件面板中的 Dialog 页标签)，并将 Name 属性设为 OpenDialog。接着，再判断用户所指定的文件是否存在，如果文件存在则创建一个 ChildForm 对象，并且设置其 Caption 属性为用户所指定的文件名称即可，所以请您在 MainForm 对象中的 MainMenu“文件/打开”选项的 onClick 事件处理过程中加入以下的代码：

“文件/打开”选项的 onClick 事件处理过程

```
procedure TMainForm.OpenFileClick(Sender: TObject);
var
  //声明一个为 TChildForm 类型的变量，而 TChildForm 类型
  //则是子窗口，其定义在 ChildUnit 单元文件中
  Child: TChildForm;
begin
  //利用 OpenDialog 对话框，打开
  if OpenDialog.Execute then
  begin
```

```
    //创建一个新的MDI子窗口(MDI child window)
    Child := TChildForm.Create(Application);

    //设置这个子窗口的Caption属性
    Child.Caption := OpenDialog.FileName;

    //并将文件的内容加载
    Child.RichEdit.Lines.LoadFromFile(OpenDialog.FileName);
  end;
end;
```

3. 另存为

这里所要介绍的功能为“另存为”。这个功能需要用到一个 TSaveDialog 对象(位于组件面板的 Dialog 页标签中)，所以请先加入一个 TSaveDialog 对象到 MainForm 对象上。并在 MainForm 菜单“另存为”的 onClick 事件处理过程中加入以下代码：

“另存为”的 onClick 事件处理过程

```
procedure TmainForm.SaveAsFileClick(Sender: TObject);
begin
   //先判断目前是否有活动中的子窗口
  if ActiveMDIChild <> nil then
    begin
      //设置SaveDialog
      SaveDialog.Title := '另存为';
      SaveDialog.DefaultExt:='RTF';
      SaveDialog.filename:=(ActiveMDIChild as TchildForm).Caption;
      if SaveDialog.Execute then
         with (ActiveMDIChild as TChildForm)do
            begin
               //将文件用新文件名加以保存
               RichEdit.Lines.SaveToFile(SaveDialog.FileName);
               //改变子窗口的Caption属性
               caption:= SaveDialog.FileName;
            end;
    end;
end;
```

4. 保存

这里所要介绍的功能为“保存”。这个功能需要用到一个 TSaveDialog 对象，但可以共享上一步骤所加入的 TSaveDialog 对象。请在 MainForm 菜单“保存”的 onClick 事件处理过程中加入以下的代码：

“保存”的 onClick 事件处理过程

```
procedure TMainForm.SaveFileClick(Sender: TObject);
begin
```

```
    //先判断目前是否有活动中的子窗口
    if ActiveMDIChild <> nil then
      begin
        //先判断指定的文件名是否存在
        if FileExists(ActiveMDIChild.Caption)then
            //若文件存在则直接保存
           (ActiveMDIChild as TChildForm ).RichEdit.Lines.

SaveToFile(Caption)
        else
            //否则，则调用新文件
            SaveAsFileClick(nil);
      end;
end;
```



由于新建或打开都必须引用 ChildUnit，因此别忘了在 MainUnit 中要引用 ChildUnit 喔！

```
Uses ChildUnit;
```

16.3.4 创建其他的窗口(如 AboutBox 窗口)

在 MDI 应用程序中，同样可以创建 AboutBox 窗口，由于在前面几个章节曾经示范如何利用对象库快速创建 AboutBox 窗口了，因此在这里不想再重复了。事实上，在 AboutBox 窗口中除了对应用程序版权的说明外，更可以显示出目前用户计算机系统的状态。本小节将为大家示范如何通过调用 Windows API 来完成 E-Mail、URL 及 FTP 等功能。

Step 1 创建新的 TForm 对象，将 Name 属性设为 AboutForm、BorderStyle 属性设为 bsDialog，并将单元文件储存为 AboutUnit。

Step 2 在 AboutForm 上加入六个 TLabel 对象，其设置如表 16.5 所示。

表 16.5

Name 属性	其他设置
Label1	Caption 设为“作者 Email：”
	Font.Size 设为 12
Label2	Caption 设为“参考地址：”
	Font.Size 设为 12
Label3	Caption 设为“FTP 站点：”
	Font.Size 设为 12
LabelEMail	Caption 设为“carlos1@ms22.hinet.net”
	Font.Size 设为 12
	Font.Color 设为 Blue
	Font.Style.fsUnderline 设为 True
	Cursor 设为 crHandPoint

续上表

Name 属性	其他设置
LabelURL	Caption 设为 “http://www.yuntech.edu.tw”
	Font.Size 设为 12
	Font.Color 设为 Blue
	Font.Style.fsUnderline 设为 True
	Cursor 设为 crHandPoint
LabelFTP	Caption 设为 “ftp://nctucca.edu.tw”
	Font.Size 设为 12
	Font.Color 设为 Blue
	Font.Style.fsUnderline 设为 True
	Cursor 设为 crHandPoint

STEP 3 请在 LabelEMail 对象的 onClick 事件处理过程加入以下的代码，这里主要是调用 Windows API 来运行发信功能。因为 ShellExecute 函数被定义于 Shellapi 单元文件中，所以您必须在 uses 子句中加入 Shellapi 单元文件。

```
procedure TAboutForm.LabelEmailClick(Sender: TObject);
begin
   ShellExecute(handle, 'open', pchar('mailto:' +
            LabelEmail.Caption), Nil, Nil, SW_SHOWNORMAL);
end;
```

STEP 4 请在 LabelURL 对象的 onClick 事件处理过程加入以下的代码，这里主要是调用 Windows API 来浏览网站。

```
procedure TAboutForm.LabelURLClick(Sender: TObject);
begin
   ShellExecute(handle, 'open', pchar(LabelURL.Caption), Nil,
                Nil, SW_SHOWNORMAL);
end;
```

STEP 5 请在 LabelFTP 对象的 onClick 事件处理过程加入以下的代码，这里主要是调用 Windows API 来存取 FTP 站点。

```
procedure TAboutForm.LabelFTPClick(Sender: TObject);
begin
   ShellExecute(handle, 'open', pchar(LabelFTP.Caption), Nil,
                Nil, SW_SHOWNORMAL);
end;
```

STEP 6 在 AboutForm 对象中加入一个 TButton 对象，并在其 onClick 事件处理过程中加入以下的代码，以结束 AboutForm。

```
procedure TAboutForm.Button1Click(Sender: TObject);
begin
```

```
    Close;
end;
```

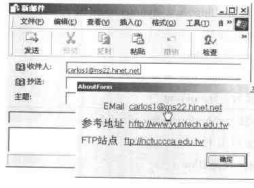


图 16.18　调用 Windows API 运行 Email、WWW 及 FTP 等功能

注意

这个 AboutForm 主要示范如何通过 Windows API 来快速运行您所想要的功能。当工程运行后，您可以打开 AboutForm，并在 Email、URL 及 FTP 等字样上按下鼠标左键，此时系统会自动打开所默认的电子邮件软件及浏览器（见图 16.18）。以下就 ShellExecute 这个 Windows API 加以说明。您可以从 Delphi 6 下拉式菜单的 Help/Windows SDK 中找到它，其原型声明如下：

```
HINSTANCE ShellExecute(
    HWND hwnd,
    LPCTSTR lpOperation,
    LPCTSTR lpFile,
    LPCTSTR lpParameters,
    LPCTSTR lpDirectory,
    INT nShowCmd
  );
```

以下就 ShellExecute API 的各项参数加以说明之：

- *hwnd* 参数

hwnd 参数代表一个指定的父窗口(parent window)，我们通常以 Appliction 的 handle 属性值作为参数。

- *lpOperation* 参数

lpOperation 参数代表操作方式，这里包括打开(open)、打印(print)及浏览(explore)等。

- *lpFile* 参数

lpFile 参数代表文件名称，其数据类型为 null-terminated string，因此，我们通常会使用 PChar 函数将一般 String 的数据类型加以转换为 null-terminated string。

- *lpParameters* 参数

lpParameters 参数代表程序运行时所需的参数，其数据类型同样为 null-terminated string。

- *lpDirectory* 参数

lpDirectory 参数代表程序所在的目录名称，其数据类型同样为 null-terminated string。

- *nShowCmd* 参数

nShowCmd 参数代表应用程序显示的类型，这些类型包括：SW_HIDE、SW_MAXIMIZE、SW_MINIMIZE、SW_RESTORE、SW_SHOW、SW_SHOWDEFAULT、SW_SHOWMAXIMIZED、SW_SHOWMINIMIZED、SW_SHOWMINNOACTIVE、SW_ SHOWNA、SW_SHOWNOACTIVATE 及 SW_SHOWNORMAL。其中 SW_HIDE 代表隐藏、SW_MAXIMIZE 代表最大化、SW_MINIMIZE 代表最小化和 SW_SHOWNORMAL 的一般显示大小等。

第 17 章

Delphi 6 数据库程序设计(一)

本章目标

本章将进入程序设计另一个重要的课题——数据库。有人说，企业界所使用的应用程序几乎有百分之九十九都离不开数据库的应用。其实这样说是一点都不为过的。因为，如果一堆数据没有更好地运用，它永远是“数据”而已；反之，如果适当的利用数据，它就能成为有用的“信息”，进而提高企业的竞争力，让企业立于不败之地。因此，您千万不能错过接下来的几个章节。本章先来谈谈数据库的概念、Delphi 6 与数据库环境的设置、通过向导创建数据库应用程序及数据库查询语言 SQL(Structured Query Language)等。

本章主题

17.1 数据库简介

17.2 Delphi 6 与数据库环境设置

17.3 Database Desktop

17.4 利用向导创建数据库程序

17.5 数据库查询语言 SQL(Structured Query Language)

17.1 数据库简介

目前大家所熟知的 Microsoft SQL Server 及 Oracle 等数据库都是关系型数据库(Relational Database System)，因此简单地说，数据库(Database)是一群相关数据的集合体。数据库是由零个或多个以上的表(Table)所组成的，每个表包含多个字段，每个字段是数据访问的最小单位(如图 17.1 所示)。以学校而言，我们可能会用一个数据库来保存学生基本数据及课程数据。而其中的“学生基本数据”、“课程数据”及“先修课程”便是所谓的表，以“学生基本数据”表而言，Name、Number、Class 及 Major 等便是用来保存学生基本数据的字段。

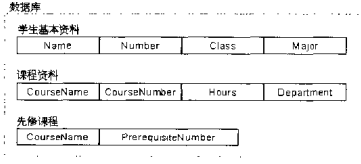


图 17.1 数据库的概念

17.2 Delphi 6 与数据库环境设置

有了数据库的简单概念之后，我们来谈谈 Delphi 6 是如何与数据库沟通的。Delphi 6 所使用的数据库可以分为“单机数据库”与“远程数据库”两大类。前者我们通常是指位于个人计算机上的本地数据库(Local Database)，例如 dBase、Access、FoxPro 或 Paradox 等。后者，我们通常是指较大型的数据库，例如 Oracle、MS SQL Server、Sybase 及 Informix 等。其结构图如图 17.2 所示。

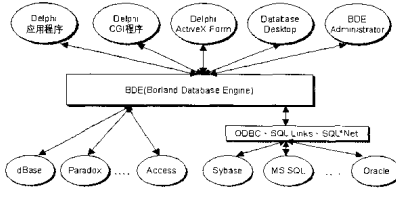


图 17.2 Delphi 6 与数据库的关系

从图 17.2 中可以看出 BDE(Borland Database Engine)是整个 Delphi 6 数据库控制的核心，换言之，BDE 是 Delphi 6 访问各种数据库的重要工具。无论您是利用 Delphi 6 所开发的应用程序、CGI 程序及 ActiveX Form 的因特网程序都可以通过 BDE(Borland Database Engine)来访问本地数据库或远程大型数据库。除此之外，您也可以利用 Delphi 6 所提供的 Database Desktop 及 BDE 工具程序直接来访问数据库。

一般来说，开发一个 Delphi 6 数据库应用程序大致上可以分为“数据库环境设置”、“数据库表(Table)设计”及“程序开发”等三个步骤。“数据库环境设置”主要是用来决定应用程序与数据库的沟通方式，关于“数据库环境设置”我们可以通过 BDE Administrator 来完成，详细说明请参考第 17.3 节。“数据库表设计”主要是用来决定应用程序所使用数据表(Table)的结构设计，我们可以通过 Database Desktop 来完成，详细说明请参考第 17.2.2 节。至于“程序开发”所含盖的范围几乎包含整本书了，但有关数据库方面的程序设计，您可以参考以后的章节，将会有详细的说明。

17.2.1　BDE Administrator

BDE Administrator 主要是用来完成数据库应用程序开发的第一步。在这个步骤中，我们有两件非常重要的事要完成。第一件事就是创建“数据库驱动程序(Driver)”，通常同一种数据库只需要一个驱动程序(Driver)就可以了，Delphi 6 为了节省大家设置的时间，已经事先建了很多常用的驱动程序。所以您不用自行去创建这些 Driver。如果您想检查这些设置，可以选择“开始/程序/Borland Delphi 6/BDE Administrator”以打开“BDE Administrator”，并选择“Configuration”页标签，如图 17.3 所示。

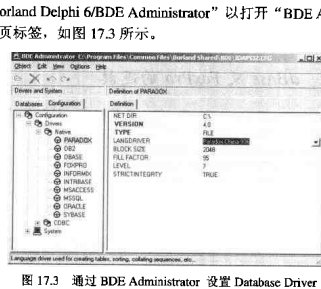


图 17.3　通过 BDE Administrator 设置 Database Driver

在数据库驱动程序(Driver)的设置中，如果您需要显示中文数据时，您必须在 LangDriver 参数中指定数据库所采用的语言，以 dBase 类的数据库而言，您必须将 LangDriver 设置“dBASE CHS cp936”、而 Paradox 类的数据库，必须将 LangDriver 设置“Paradox China 936”。

在本小节中，您所要完成的第二件事是创建“数据库别名(Alias)”。对同一个数据库，我们可以创建多个数据库别名，因此数据库别名也就是数据库的逻辑名称。在数据库别名中，我们可以指定此数据库所使用的“驱动程序(Driver)”，对于“本地数据库”的别名，我们可

以指定数据库的工作目录、对于“远程大型数据库”的别名，我们可以指定远程数据库的主机名称(Host Name)及用户名称(User Name)等。如果您想检查这些设置，可以选择“开始/程序/Borland Delphi 6/BDE Administrator”以打开“BDE Administrator”，并选择“Databases”页标签，如图 17.4 所示。

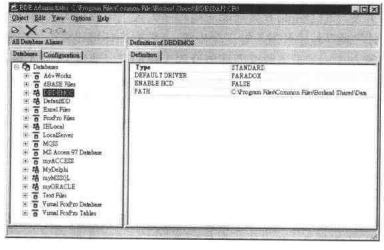


图 17.4 通过 BDE Administrator 设置 Database Alias

使用数据库别名最大好处便是可以提高程序的可移植性，如果远程数据库的主机名称或本地数据库的工作目录有任何变动，我们先前所开发的应用程序完全不需要改写，只需对数据库别名做少许的修正即可。

17.2.2 Paradox、dBase 及 FoxPro 的连接

如果您要连接 Paradox、dBase 及 FoxPro 等本地数据表，我们可以将其分为下面两个步骤，分述如下：

步骤一：创建 BDE 别名(BDE Alias)。

步骤二：连接测试。

1. 创建 BDE 别名(BDE Alias)

在 Delphi 6 中，BDE 允许您直接访问 Paradox、dBase 或 FoxPro 等区域性的数据库，其设置也很简单。首先，您必须选择“开始/程序/Borland Delphi 6/BDE Administrator”以打开“BDE Administrator”，并选择“Databases”页标签。在“Databases”页标签中按下鼠标右键，并选择“New...”，如图 17.5 所示。

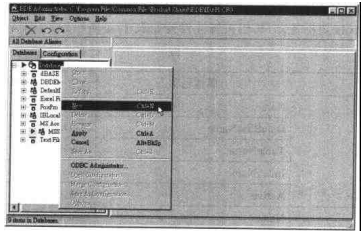


图 17.5　新建 Database Alias

接着，会出现“New Database Alias”对话框，请选择“STANDARD”。这里的“Standard”包括了 Paradox、dBase 或 FoxPro 等本地的数据库。接着您只要为这个别名取一个新名字，并且选择“数据库驱动程序(Driver)”，并为此别名指定数据库的工作目录及保存设置数据即可。在下图中主要是示范创建一个 MyDelphi 别名，其默认驱动程序(DefaultDirver)设为 dBase，而且其工作目录(Path)设为“C:\Program Files\Common Files\Borland Shared\Data”(这个目录也是 Delphi 6 的范例数据表保存的地方)，如图 17.6 所示。

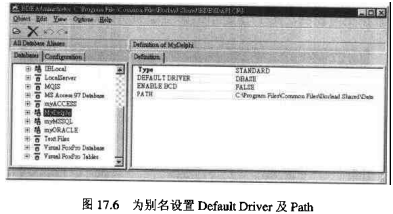


图 17.6　为别名设置 Default Driver 及 Path

2. 连接测试

经过上面的努力您已经完成了基本的设置，接下来便是来测试一下是否连接成功。

范例 17-1

目的：本范例主要说明如何创建本地数据库程序。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入 TTable 对象(位于组件面板的 BDE 页标签)、TDataSource(位于

对象模版上的 DataAccess 页标签)、TDBGrid 及 TDBNavigator(位于组件面板的 DataControls 页标签)，其相关属性设置如表 17.1 所示。

表 17.1

对象	属性名称	属性设置值
TTable	Name	Table1
	DataBaseName	MyDelphi
	TableName	employee.db
	IndexName	ByName
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1
	VisibleButtons	(决定出现的按钮)

步骤 3 保存工程。

步骤 4 按【F9】键运行工程。工程运行后，您可以利用 TDBNavigator 对象来移动数据指针或是数据更新控制。图 17.7 是示范如何添加一条数据。

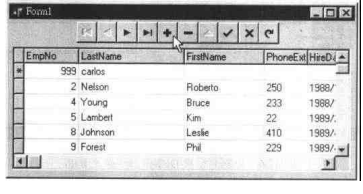


图 17.7 本地数据表连接测试结果

17.2.3 MS SQL Server 的连接

如果您要连接 Microsoft SQL 数据库，我们可以将其分为下面几个步骤，分述如下：

步骤一：安装客户端软件。

步骤二：创建 Microsoft SQL Server 别名 Server Alias。

步骤三：创建 BDE 别名 BDE Alias。

步骤四：连接测试。

1. 安装客户端软件

如果您要编写 Microsoft SQL Server 的 Client / Server 数据库应用程序，那么您必须在用户端(或称客户端)安装 Microsoft SQL Server 客户端软件。因此本小节将示范如何安装 Microsoft SQL Server 2000 的客户端软件，至于服务端的安装步骤请参考相关的书籍。

(1) 请运行 Microsoft SQL Server 2000 安装程序，这里以安装 Windows 2000 客户端为例。如果您是使用 Windows 95/98/ME 的操作系统时，系统会提醒您在该操作系统下只能安装客户端软件，而无法安装服务器的组件，请单击“确定”按钮，如图 17.8 所示。



图 17.8　运行 Microsoft SQL Server 2000 安装程序

(2) 接下来是一个欢迎使用的画面，您只要单击“下一步”按钮即可，如图 17.9 所示。

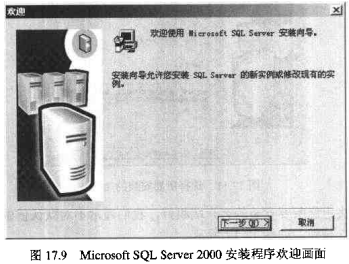


图 17.9　Microsoft SQL Server 2000 安装程序欢迎画面

(3) 接下来是一个许可协议说明画面，如果您是合法的用户，请单击“是”按钮，如图 17.10 所示。

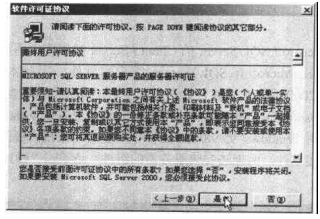


图 17.10 Microsoft SQL Server 2000 安装程序许可协议说明

步骤 4 选择所要安装的功能。其中“仅客户端工具”代表安装客户端相关工具与软件；“服务器和客户端工具”代表安装数据库服务器软件；Connectivity Only 代表只安装必要的驱动程序而已。这里以“仅客户端工具”为例，如图 17.11 所示。



图 17.11 选择所要安装的功能

步骤 5 接着，请决定您所要安装的客户端组件，我们通常不对默认值做任何的修改，单击“下一步”按钮，如图 17.12 所示。

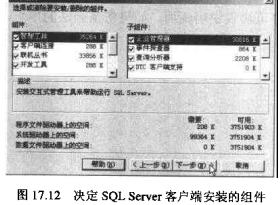


图 17.12 决定 SQL Server 客户端安装的组件

⑥ 系统在开始复制文件前会再作一次确认，如果您确定要安装，请单击“下一步”按钮继续，如图 17.13 所示。

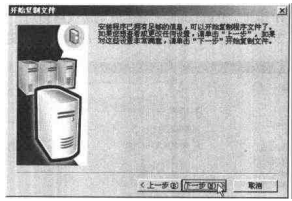


图 17.13　确认安装 SQL Server 客户端组件

⑦ 经过一段时间的复制后，如果系统出现如图 17.14 所示的画面，那表示已经安装完成，您只要单击“完成”按钮，客户端的安装即算完成。



图 17.14　安装 SQL Server 客户端完成

2. 创建 Microsoft SQL Server 别名(Server Alias)

安装 Microsoft SQL Server 客户端软件后，接着便是设置 Microsoft SQL Server 别名，也就是所谓的 Server Alias。为什么需要创建 Server Alias 呢?因为创建 Server Alias 可以提高应用程序的可移植性。当应用程序需要更改连接的数据库服务器时，您只要修改 Server Alias 就可以，而不需要修改 BDE Alias，更不需要修改程序的内容。以下示范如何创建 Server Alias。

① 从“开始/程序/Microsoft SQL Server /客户端网络实用工具”打开“SQL Server 客户端网络实用工具”。并在“General”页标签中选择所使用的网络通讯协议(default network)与其使用的顺序。关于网络通讯协议您可以根据实际需要而作适当的调整，这里以优先使用 TCP/IP 通讯协议为例。

② 接着您在“Alias”页标签中单击“Add”钮以添加一个 Server alias。在“Add Network Library Configuration”对话框中输入 Server alias、Network libraries、Computer Name

及 Port number 等数据。Server alias 是指您自己所设置的别名，其与数据库服务器名称无关。Network libraries 是指通讯协议，这里以 TCP/IP 为例。Computer Name 是指数据库服务器所在的计算机名称，通常数据库服务器与计算机名称相同，当然您也可以设置其 IP 地址。Port number 是供通讯协议所使用，通常使用默认值(1433)，在 SQL Server 2000 以后允许您动态决定 Port number，如图 17.15 所示。

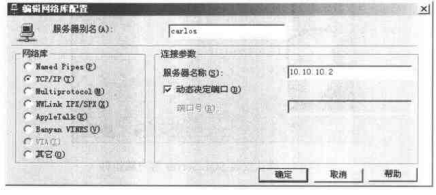


图 17.15　设置 Server Alias 相关设置

步骤 3　如果您在上一个步骤已经正确设置 Server Alias，并单击“OK”按钮，那么系统会将这些设置数据加入 Server alias configurations 中，也代表该 Server Alias 已创建完成，最后请单击“确定”按钮，如图 17.16 所示。

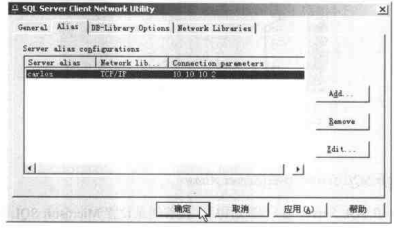


图 17.16　Server Alias 创建完成

步骤 4　如果想测试 Server Alias 是否正确，那么您可以使用“查询分析器”。请从“开始/程序/Microsoft SQL Server / 查询分析器”打开“查询分析器”。打开后，请选择前面所创建的“Server Alias”，这里以 carlos 为例。并输入用户名称与密码，这里以 sa 为例，最后再单击“OK”按钮，如图 17.17 所示。



图 17.17　登录数据库服务器

5 如果您所创建的 Server Alias 正确的话，那么您可以在查询分析器中选择使用的数据库(Database)及运行 SQL 命令，如图 17.18 所示。

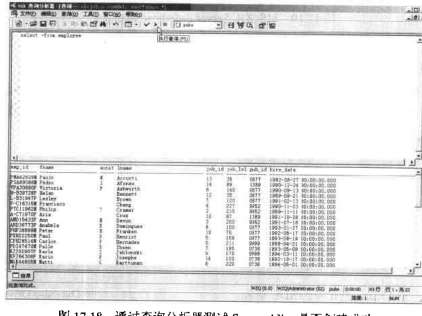


图 17.18　通过查询分析器测试 Server Alias 是否创建成功

3. 创建 BDE 别名(BDE Alias)

MS SQL Server 目前最新的版本是 2000，因此我们在此以 MS SQL Server 2000 作示范。在创建 BDE 别名之前，我们先说明一下 MS SQL Server 中的默认表(Table)。您可选择“开始/程序/MS SQL Server / 企业管理器”打开“SQL Server Enterprise Manager”。在“Enterprise Manager”中我们可打开“数据库”文件夹，并点选“pubs”数据库，这个数据库就是待会我们要连接的数据库。这个数据库中包含很多的数据表，例如 Authors、Discounts、employee 等，如图 17.19 所示。

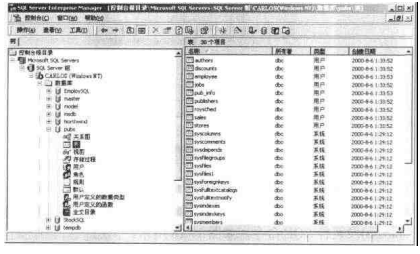


图 17.19 MS SQL Server Enterprise Manager

因为设置 BDE 别名之前最好有这些概念，如此一来您才可以举一反三。以下数个步骤将示范如何创建 BDE Alias。

Step 1 从“打开/程序/Borland Delphi 6/ BDE Administrator”中打开“BDE Administrator”。打开后，按右键新建一个别名，请选择“MSSQL”，如图 17.20 和图 17.21 所示。

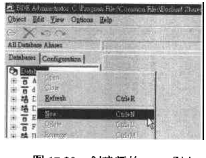


图 17.20 创建新的 BDE 别名

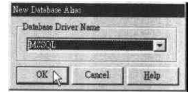


图 17.21 选择 Database Driver Name

Step 2 接着为这个新的数据库别名(Database Alias)的参数设置新值，这些参数如表 17.2 所示。

表 17.2

参数名称	参数设置值
Database Alias	MyMSSQL
DATABASE NAME	Pubs(Database Name)
SERVER NAME	carlos(Server Alias)
USER NAME	sa(用户名称)

Step 3 参数设置之后，请单击“Apply”按钮，如图 17.22 所示。

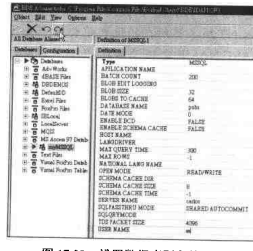


图 17.22　设置数据库别名的参数

4 此时 BDE Administrator 会确认您是否保存此 BDE Alias 的设置数据，如果您设置数据正确，请单击“OK”按钮，如图 17.23 所示。

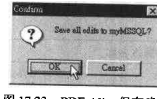


图 17.23　BDE Alias 保存确认

5 如果您想测试 BDE Alias 是否成功创建，请您在 BDE Alias 名称上双击两下鼠标右键 (这里以 myMSSQL 为例)。当您快按两下后，系统会出现“Database Login”对话框，您只要输入正确的用户帐号与密码并单击“OK”按钮即可，如图 17.24 所示。如果没有出现任何的错误信息，而且设置信息也呈粗黑体显示，那即表示此 BDE Alias 已成功地登录数据库服务器，如图 17.25 所示。

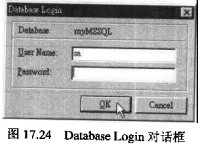


图 17.24　Database Login 对话框

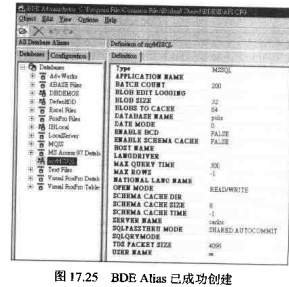


图 17.25　BDE Alias 已成功创建



如果您不想创建 Server Alias，那么您也可以只在 BDE Alias 中直接指定 HOST NAME(Windows NT 服务器所在的位置或名称)、SEVER NAME(MS SQL Server 名称)、DATABASE NAME(数据库名称)及 USER NAME(用户名称)也可以直接连上 Microsoft SQL Server，但笔者并不建议您这样使用。

4. 连接测试

经过上面的努力您已经完成了基本的设置，接下来我们将写一个简单的程序来测试一下是否可以正确从数据库服务器中获取数据。



目的：本范例主要示范如何连接 MS SQL Server。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象上加入一个 TDatabase 对象、一个 TTable 对象、一个 TDataSource 对象、一个 TDBNavigator 对象及一个 TDBGrid 对象。其相关属性设置如表 17.3 所示。

表 17.3

对象	属性名称	属性设置值
TDatabase	AliasName	MyMSSQL
	Name	Database1
	DatabaseName	TestDB
	HandleShared	True
	LoginPrompt	False
	Params	UserName=sa Password= (请视您在 MS SQL Server 的权限而定)
	Connected	True
TTable	Name	Table1
	DatabaseName	TestDB
	TableName	dbo.employee
	IndexName	PK_emp_id
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1

3 保存工程。

4 按【F9】键运行工程。工程运行后，您可以利用 TDBNavigator 对象对此指定数据表进行各种的处理，如图 17.26 所示。

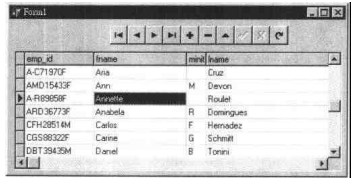


图 17.26　MS SQL 数据库连接测试

17.2.4　Oracle Server 的连接

如果您要连接 Oracle 数据库，我们可以将其分为下面几个步骤，分述如下：

步骤一：安装客户端软件(Oracle Client)。

步骤二：创建数据库别名(Database Alias)。

步骤三：创建 BDE 别名(BDE Alias)。

步骤四：连接测试。

1. 安装 Oracle Client(客户端软件)

Oracle 数据库目前的最新版本 9i，但是笔者手边只有 8.1.7，也就是所谓的 Oracle 8i，因此本书将以 Oracle 8i Enterprise Edition 为例。以下将以数个步骤说明如何安装 Oracle 客户端软件。

1 请在 Oracle 8i Enterprise Edition 安装光盘的根目录下运行“Setup.exe”，打开“Oracle universal Installer”。在欢迎画面中请单击“下一步”按钮，如图 17.27 所示。



图 17.27　Oracle universal Installer 欢迎画面

步骤 2 请决定文件保存的目的位置，这里以“C:\Oracle\Ora81”为例，最后请单击“下一步”按钮，如图 17.28 所示。

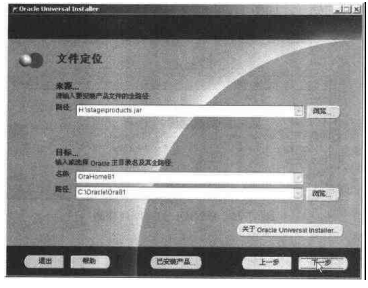


图 17.28 决定文件保存的目的位置

步骤 3 选取所要安装的产品，这里以“Oracle8i Client 8.1.7.00”为例，最后请单击“下一步”按钮，如图 17.29 所示。

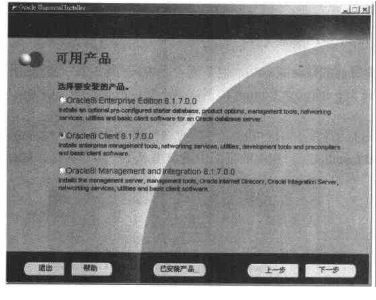


图 17.29 选取所要安装的产品

步骤 4 决定所要安装的类型，这里以“Application User”为例，最后请单击“下一步”按钮，如图 17.30 所示。

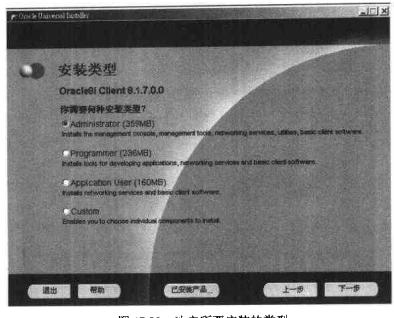


图 17.30　决定所要安装的类型

⑤ 系统会以摘要的方式显示您所选取的安装项目，如果正确，请单击“安装”按钮，如图 17.31 所示。

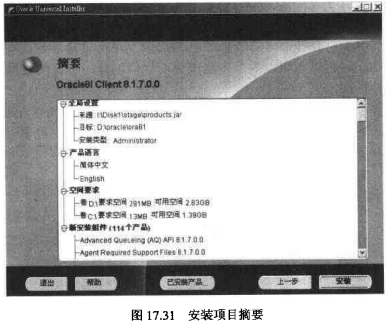


图 17.31　安装项目摘要

⑥ 经过一段时间的复制文件，安装即告完成。由于这是一个由 JAVA 所开发的安装程序，如果您的计算机不是很快，那么文件复制可能需要一段时间，所以请您待心等候，笔者就曾经以为死机，因而关机好几次，如图 17.32 所示。

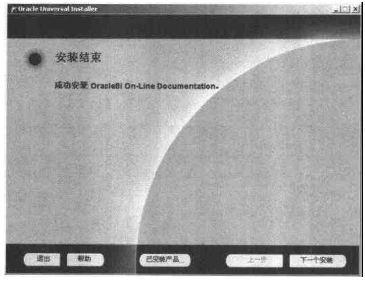


图 17.32 结束安装

2. 创建数据库别名(Database Alias)

如果您已经正确的安装 Oracle 客户端软件，接下来便是创建一个新的数据库别名(Database Alias)。我们可以利用 Oracle 所提供的“Net8 Assistant”工具程序来创建数据库别名，这里的数据库别名是真正连接到数据库的别名，与下一步骤的 BDE 别名(Borland Database Engine Alias)是有些不同的。换言之，BDE 别名必须通过数据库别名才能连接到 Oracle Database Server。以下将以数个步骤来说明如何创建一个新的数据库别名(Database Alias)。

步骤 1 从“开始/程序/Oracle - OraHome81/Network Administration /Net8 Assistant”打开“Net8 Assistant”工具程序。在 Net8 Assistant 中，请先选择“服务命名”项目，再从下拉菜单选择“创建...”，如图 17.33 所示。

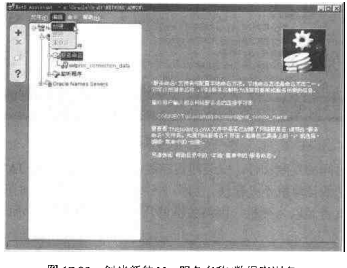


图 17.33 创建新的 Net 服务名称(数据库别名)

(2) 决定新的 Net 服务的名称，这里以 Oracle8i 为例，如图 17.34 所示。

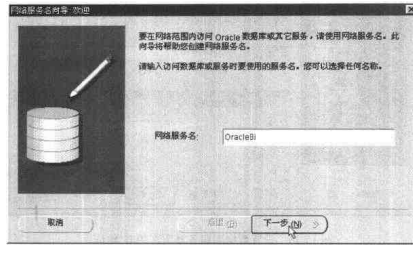


图 17.34　决定 Net 服务名称

(3) 决定 Net 服务名称所使用的通讯协议，这里以 TCP/IP(Internet 协议)为例，请单击“Next”按钮，如图 17.35 所示。

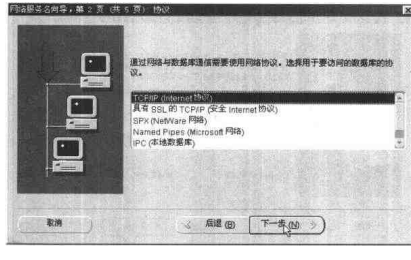


图 17.35　选择通讯协议

(4) 输入数据库所在的 IP 地址或计算机名称及连接端口号，这里 IP 地址以“10.10.10.2”为例，而连接端口的默认值为“1521”。最后，请单击“下一步”按钮，如图 17.36 所示。

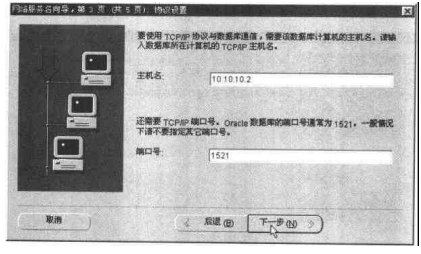


图 17.36　设置数据库服务器所在的 IP 地址

(5) 接着，请输入数据库服务器的 Net 服务名称。因为笔者所连接的数据库版本为 Oracle 8i，因此可以输入服务名称。如果您所连接的数据库为 Oracle 8i 以前的版本，请使用 SID(System Identifier)。关于 Oracle 8i 的服务名称与 SID 您可以查询数据库管理员。输入后，请单击“下 一步”按钮，如图 17.37 所示。

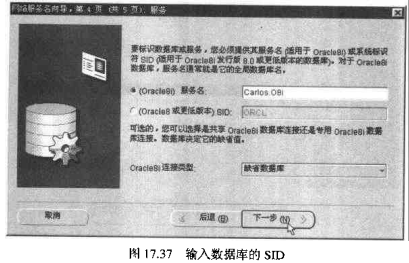


图 17.37　输入数据库的 SID

(6) 请单击“测试”按钮以测试目前所创建的数据库别名是否可以成功的连接，如图 17.38 所示。

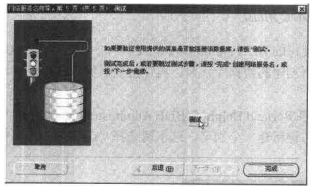


图 17.38　测试数据库别名是否可以成功连接

7 如果连接测试成功，系统会出现已顺利完成连接测试的信息，您只要单击“关闭”按钮即可。如果无法成功连接，您可能要询问数据库管理员有关数据库的相关设置，如图 17.39 所示。

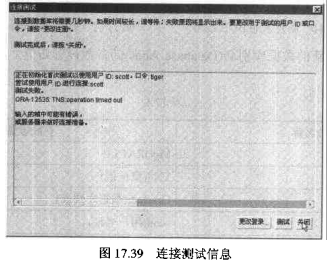


图 17.39　连接测试信息

8 如果已经成功的完成连接测试，您可以将这些设置数据加以保存，如图 17.40 所示。

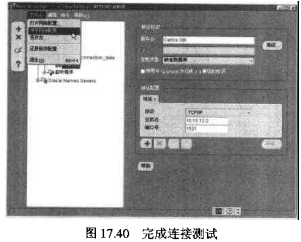


图 17.40　完成连接测试

3. 创建 BDE 别名(BDE Alias)

利用 Delphi 6 所提供的 BDE Administrator 工具程序，创建一个 BDE 别名。这里的别名是用来连接上一步骤所创建的数据库别名，再连接到数据库。以下将以数个步骤来说明如何创建 BDE 别名。

步骤 1 从“打开/程序/Borland Delphi 6/ BDE Administrator”中打开“BDE Administrator”。打开后，按右键新建一个别名，这里请选择“ORACLE”，如图 17.41 和图 17.42 所示。

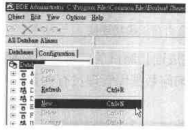


图 17.41 创建新的 BDE 别名



图 17.42 选择 Database Driver Name

步骤 2 接着为这个新的数据库别名(Database Alias)的参数设置新值，这些参数如表 17.4 所示。

表 17.4

参数名称	参数设置值
Database Alias	MyORACLE
NET PROTOCOL	TNS(默认值)
SERVER NAME	Oracle8i(数据库别名)
USER NAME	scott(用户名称)

步骤 3 参数设置之后，请单击“Apply”按钮，如图 17.43 所示。

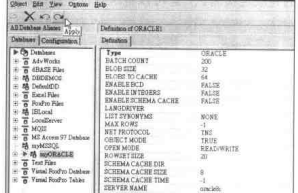


图 17.43 设置数据库别名的参数

步骤 4　此时 BDE Administrator 会确认您是否保存此 BDE Alias 的设置数据，如果您设置数据正确，请单击“OK”按钮，如图 17.44 所示。

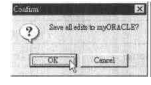


图 17.44　BDE Alias 保存确认

步骤 5　如果您想测试 BDE Alias 是否成功创建，请您在 BDE Alias 名称上双击两下鼠标右键（这里以 myORACLE 为例）。当您快按两下后，系统会出现“Database Login”对话框，您只要输入正确的用户帐号与密码并单击“OK”按钮即可，如图 17.45 所示。这里以“scott”为用户帐号，而以“tiger”为密码。如果没有出现任何的错误信息，而且设置信息也呈粗黑体显示，那即表示此 BDE Alias 已成功地登录数据库服务器，如图 17.46 所示。

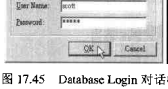


图 17.45　Database Login 对话框



图 17.46　BDE Alias 已成功创建

4. 连接测试

经过上面的努力您已经完成了基本的设置，接下来我们将写一个简单的程序来测试一下是否可以正确从数据库服务器中获取数据。

范例 17-3

目的：本范例主要示范如何连接 Oracle Database Server。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象上加入一个 TDatabase 对象、一个 TTable 对象、一个 TDataSource 对象、一个 TDBNavigator 对象及一个 TDBGrid 对象。其相关属性设置如表 17.5 所示。

表 17.5

对象	属性名称	属性设置值
TDatabase	AliasName	MyOracle
	Name	Database1
	DatabaseName	TestDB
	HandleShared	True
	LoginPrompt	False
	Params	UserName=Scott Password=tiger
	Connected	True
TTable	Name	Table1
	DatabaseName	TestDB
	TableName	EMP
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1

③ 保存工程，并按【F9】功能键运行工程。运行工程后，您可以利用 TDBNavigator 对象对此指定数据表进行各种的处理，如图 17.47 所示。

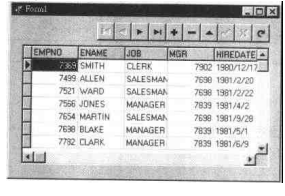


图 17.47 Oracle 数据库连接测试

17.2.5 以 ODBC 连接 Access 数据库

如果您要通过 ODBC 连接数据库，我们可以将其分为下面几个步骤，分述如下：

步骤一：创建 ODBC 的 DSN(Data Source Name)。

步骤二：连接测试。

1. 创建 ODBC 的 DSN(Data Source Name)

以下数个步骤将示范如何创建一个 ODBC 的 DSN，以访问 ACCESS 数据库。

① 双击控制面板的“数据源 ODBC”图标，打开“ODBC Data Source Administrator”。点选“System DSN”页标签，并单击“Add”按钮，如图 17.48 所示。



图 17.48　ODBC Data Source Administrator

② 选择“Microsoft Access Driver”，并单击“完成”按钮，如图 17.49 所示。

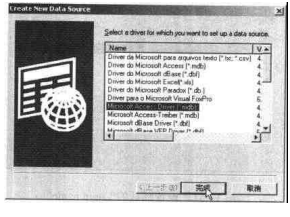


图 17.49　选择 Microsoft Access Driver

③ 此时系统会出现“ODBC Microsoft access Setup”对话框。请您在“数据源名”输入“myACCESS”、“说明”输入“通过 ODBC 访问 ACCESS 数据库”。并单击“选择”按钮选择保存 Access 数据库的实际目录，这里以“C:\ Program Files\Common Files\Borland Shared\Data\dbdemos.mdb”为例。设置完毕后，单击“高级”按钮进行其他相关设置，如图 17.50 所示。

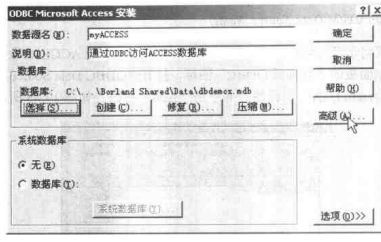


图 17.50　设置 DSN 的实际目录路径

(4) 在“设置高级选项”对话框中请设置“登录名称”为“admin”，并单击“确定”按钮，如图 17.51 所示。



图 17.51　设置默认的 Login name

(5) 最后再单击“确定”按钮，DSN 设置完成。

(6) 当您设置好 ODBC 的 DSN 后，您可以从“打开/程序/Borland Delphi 6/ BDE Administrator”中打开“BDE Administrator”来查看 ODBC 的内容。在 BDE Administrator 中只要是 这样的图标就是代表 ODBC 所创建的 DSN，因此您同样可以在图标的上方双击，以查看是否可以正确的连接，如图 17.52 所示。

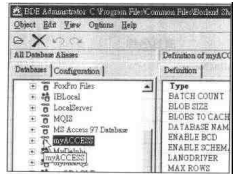


图 17.52　查看 ODBC 的 DSN 是否正确

7 当您双击后，系统会出现“Database Login”对话框，您只要输入正确的用户帐号与密码并单击“OK”按钮即可，如图 17.53 所示。如果没有出现任何的错误信息，而且设置信息也呈粗黑体显示，那即表示此 BDE Alias 已成功地登录数据库服务器，如图 17.54 所示。

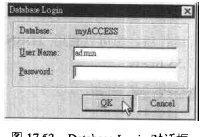


图 17.53　Database Login 对话框

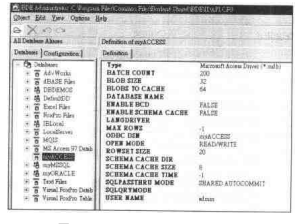


图 17.54　ODBC 的 DSN 已成功连接

2. 连接测试

经过上面的努力您已经完成了基本的设置，接下来我们将写一个简单的程序来测试一下是否可以正确从数据库服务器中获取数据。



17-4

目的：本范例主要示范如何通过 ODBC 连接 Access 数据库。

1 创建新的工程。

2 在 TForm 对象上加入一个 TDatabase 对象、一个 TTable 对象、一个 TDataSource 对象、一个 TDBNavigator 对象及一个 TDBGrid 对象。其相关属性设置如表 17.6 所示。

表 17.6

对象	属性名称	属性设置值
TDatabase	AliasName	MyACCESS
	Name	Database1
	DatabaseName	TestDB
	HandleShared	True
	LoginPrompt	False
	Connected	True
TTable	Name	Table1
	DatabaseName	TestDB
	TableName	employee
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1

3 保存工程，并按【F9】功能键运行工程。工程运行后，您可以利用 TDBNavigator 对象对此指定数据表进行各种的处理，如图 17.55 所示。

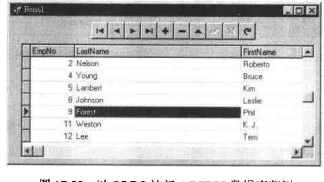


图 17.55 以 ODBC 访问 ACCESS 数据库范例

17.3 Database Desktop

Database Desktop 是 Delphi 6 所提供的数据库管理工具，我们只要配合 BDE Administrator 所创建的数据库别名，就可以利用 Database Desktop 来创建数据库的表(Table)，或是对某指定表做输入、查询、修改或删除等管理的工作。如果您要启动 Database Desktop，只要选择“开始/程序/Borland Delphi 6”并运行 Database Desktop 即可。

17.3.1 创建新的数据表(Table)

如果您要创建新的数据表，请先打开 Database Desktop，并选择“File/New/Table…”。此时会出现表类型 Table Type)的对话框，请选择“dBase for windows”。为什么选择这种表类型呢?因为这是最普遍的数据表，所以我们以它为例，当然您也可以选择其他的表类型，如图 17.56 所示。



图 17.56 选择数据表类型(Table Type)

当您选择好 Table Type，且单击“OK”按钮后，会出现“Create dBase for Windows Table”对话框。接着请您在 Field Name 字段中输入表中的字段名称，在 Type 字段中选择数据类型(您也可以在 Type 字段上按下空格键再选择适当的数据类型)，并在 Size 字段中输入数据类型的长度(如果此数据字段包含小数字数，您必须在 Dec 字段中输入小数字数)，如图 17.57 所示。

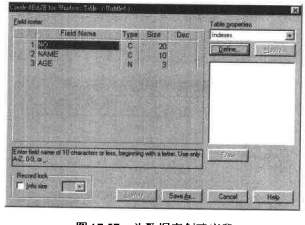


图 17.57 为数据表创建字段

当您设置完数据字段后，还有非常重要的一件事，那就是创建索引字段。什么是索引(index)字段呢?索引是一种创建在一个字段或多个字段以上的数据库对象，它以逻辑顺序来访问表上的数据。事实上，对一个数据库或数据表来说，索引并不是必要的。因为我们通过 SQL 来处理数据根本可以不需要使用它，但是如果我们使用适当的索引，可以提高查询的速度。虽然索引有以上的好处，但索引也有它的坏处，例如我们每创建一个索引，就需要另外的磁盘空间。再者，只要有任何更改数据库的内容的操作，索引文件也会一起被更改，因此其运行效率就会较差。如果您想创建索引，请在“Create dBase for Windows Table”对话框右上角选择“Indexes”，并单击“Define”按钮。此时会出现“Define Index”对话框。接着您只要点选所要创建索引的字段，并单击“OK”按钮，再输入索引的名称即可，如图 15.58 所示。

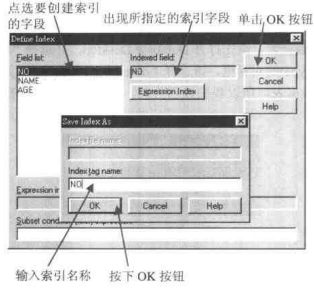


图 17.58　为数据表创建索引字段

建好索引后，按着就要将这个表数据存盘。如果您有照着步骤做，现在应该已经回到“Create dBase for Windows Table”对话框了，您只要单击“Save As..”按钮就可以存盘了。当您单击“Save As..”按钮之后，便会打开“Save Table As”对话框。此时您只要选取之后我们所创建的数据库别名，并输入 Table Name 就可以了，如图 17.59 所示。



图 17.59　保存数据表设置数据

17.3.2　为数据表(Table)添加数据

创建了数据表，接下来就是为这个数据表输入一些数据。当然，这个动作您可以通过程序来控制。但是，在介绍如何编写程序之前，我们先利用 Database Desktop 来示范如何输入数据。首先打开 Database Desktop，并利用下拉式菜单打开在上一小节所创建的数据表(Table)，如图 17.60 所示。

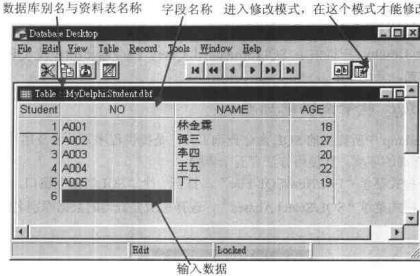


图 17.60　利用 Database Desktop 创建数据

表 17.7 就是 Database Desktop 中各项常用的功能项说明。

表 17.7

常用功能项	功能说明
	将目前的数据剪切到剪贴板
	将目前的数据复制到剪贴板
	将剪贴板中的数据贴到目前插入点的位置
	重新设置数据表(Restructure)
	将光标移到数据表的第一条
	将光标移到目前位置的前一页数据
	将光标移到目前位置的前一条数据
	将光标移到目前位置的下一条数据
	将光标移到目前位置的后一页数据
	将光标移到数据表的最后一条
	进入字段显示模式(Field View)
	进入数据修改模式(Data Edit)
Record/Insert	在目前光标所在位置之前插入一条新的数据
Record/Delete	删除光标所在的整笔数据
Tools/Utilities/Add	将某个数据表中的某条数据加到另一个数据表中
Tools/Utilities/Copy	将指定的数据表或 SQL 命令文件复制一份
Tools/Utilities/Delete	删除指定的数据表或 SQL 命令文件
Tools/Utilities/Empty	清除指定数据表的所有数据
Tools/Utilities/Rename	为指定的数据表或 SQL 命令文件重新命名
Tools/Utilities/Sort	将指定数据表的数据加以排序

17.3.3 查询数据表(Table)的数据

在 Database Desktop 中提供了 SQL(Structured Query Language)命令及 QBE(Query By Example)两种方式，分述如下。

17.3.3.1 以 SQL 命令查询数据库

Database Desktop 中所提供的 SQL 命令查询工具，是提供程序开发者及用户一个可以利用 SQL 命令来查询的环境。其使用方法有五个步骤。

1. 选择下拉式菜单“File/New/SQL File”，打开一个“SQL”命令窗口。
2. 选择下拉式菜单“SQL/Select Aliases”，选择您所要查询的数据库别名，如图 17.61 所示。



图 17.61 选择别名(Alias)

3. 在 SQL 命令窗口中输入适当的 SQL 命令。有关 SQL 命令的用法您可以参考数据库的相关书籍，或是参考本书第 18.3 节的说明，如图 17.62 所示。

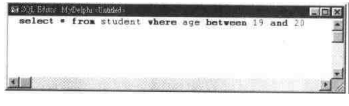


图 17.62 输入 SQL 命令

4. 选择下拉式菜单“SQL/Run SQL”运行“SQL”命令，其运行结果会打开新的窗口，以数据表的方式显示，如图 17.63 所示。



图 17.63 返回查询结果

5. 如果您想保存 SQL 命令，以选择“File/Save”保存“SQL”命令。

17.3.3.2 以 QBE 访问异构数据库

大部分的数据库都支持标准的 ANSI SQL。那什么是 QBE(Query By Example)呢?其实

QBE 是 Paradox 数据表专用的 SQL 语法，因此 BDE(Borland Database Engine)同时支持 ANSI SQL 格式及 QBE 格式。因此您可以使用 QBE 的语法来查询您所指定的数据库。除此之外，QBE 还有一个非常重要的功用，那就是“同时访问异构数据库”。那什么又是“异构数据库”呢?在一个不算太小的公司可能会有一些不同的数据库，例如生产部门可能使用 Oracle Server，而研发部门可能使用 Sysbase Server，甚至销售部门可能使用 MS SQL Server。因此，如果您想同时访问这个数据，那它们就是所谓的异构数据库。以下示范如何利用 QBE 来产生连接异构数据库。

1. 打开“Database Desktop”，并选择下拉式菜单“File/QBE Query..”，打开一个数据库表，在此以“DBDEMOS 的 animals.dbf”为例。
2. 利用下拉式菜单“Edit/Add Table”，打开另一个数据库表，在此以“MyOracle 的 Stock”为例，您可以选择其他的文件。
3. 利用鼠标点选所要显示的数据字段，如果您有点选某个数据字段，则该字段前面会出现“√”符号。
4. 点选(Join Tables)按钮，从两个数据表中指定所要 Join 的字段名称，如图 17.64 所示。



图 17.64　选择及指定 Join 字段

5. 利用下拉式菜单“Query/Show SQL”，打开 Database Top 所产生的 QBE 源代码。您只要将这些 QBE 源代码配合 TQuery 对象运行，在运行之前，您要先确定这些数据表是有效的，如图 17.65 所示。

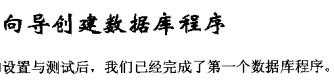


图 17.65　QBE 源代码

17.4　利用向导创建数据库程序

经过了上面的设置与测试后，我们已经完成了第一个数据库程序。当您对 Delphi 6 创建数据的步骤有一些概念之后，我们再来谈谈 Delphi 6 与数据库之间的关系，如图 17.66 所示。

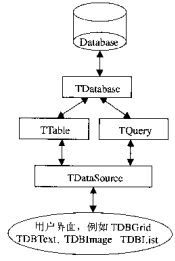


图 17.66 Delphi 与数据库的关系

从上图中可以看出用户通过我们所设计的使用界面，例如 TDBGrid、TDBText、TDBImage、TDBList 等对象从 TDataSource 对象获取数据，而 TDataSource 对象主要是将 TTable 对象与 TQuery 对象从数据库所获取的数据传给用户。因此可知一个基本的数据库程序必须具备这些基本的对象。

17.4.1 向导创建简单的数据库程序

就如同前面所说，要创建一个基本的数据库程序必须要包含 TDatabase 对象、TTable 或 TQuery 对象、TDataSource 及用户接口对象。其中的 TDatabase 对象并非必要，当您连接的是区域性数据表的话，那这个对象是可以省略的。至于 TTable 或 TQuery 对象您只要根据需要选择其中一种使用就可以，这两个对象的详细介绍请参考第 14 章。而使用接口对象可以根据需要选择适当的对象使用。看完上一小节的数据库程序的创建示范，相信您应该也有同感：利用 Delphi 6 创建数据库程序实在是太容易了。是的，除了手动创建数据库很容易外，Delphi 6 另提供数据库程序创建的向导供大家使用。以下就示范如何利用向导来创建数据库程序。

范例 17-5

目的：本范例主要示范如何利用向导创建简单的数据库程序。

STEP 1 创建新的工程。

STEP 2 选择下拉式菜单“Database/Form Wizard”，打开“Database Form Wizard”对话框，在“Form Options”中选择“Create a simple Form”，“DataSet Options”选择“Create a from using TTable Objects”，Form Option 主要是决定这个数据库应用程序的类型，如果您选择 Create a simple form，表示这个应用程序是针对单一数据表(Table)。反之，如果您选择 Create a master/detail form，表示这个应用程序是由两个数据表所组成的，而且这两个数据表成 master and detail 的关系(参考范例 17-6)。而 DataSet Options 是用来决定所使用 DataSet 的种类，这里有 TTable 与 TQuery 两种对象可供

选择。我们这里以“Create a simple form”与“Create a form using TTable objects”选择了适当的设置之后，请单击“Next”按钮，如图 17.67 所示。

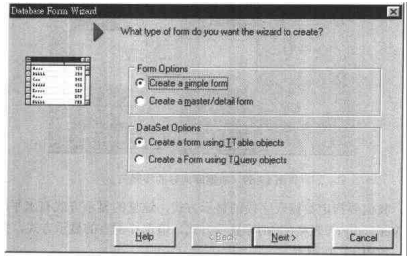


图 17.67　数据库向导的步骤一

3 接着，要指定数据库表(Table)的位置，我们以 Delphi 6 所默认的数据表为例，这些数据表在“C:\Program Files\Common Files\Borland Shared\Data”的目录下。我们选择“animals.dbf”，并单击“Next”按钮，如图 17.68 所示。

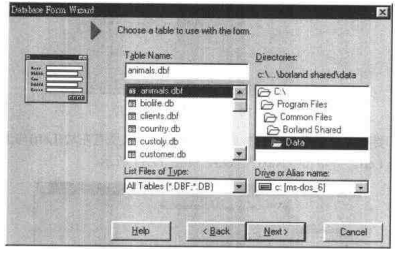


图 17.68　数据库向导的步骤二

4 接着，我们要指定所要显示的数据表字段。您可以利用上、下按键，指定字段出现的顺序，这里请选择所有的字段。并单击“Next”按钮，如图 17.69 所示。

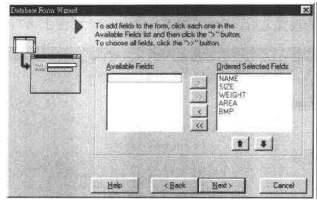


图 17.69　数据库向导的步骤三

STEP 5 接着，我们要指定数据表字段的显示方式，这里的显示方式有水平(Horizontally)、垂直(Vertically)与表格(Grid)等显示方式。请选择垂直的显示方式，并单击“Next”按钮，如图 17.70 所示。

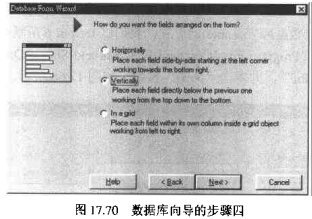


图 17.70　数据库向导的步骤四

STEP 6 接着，要决定每个数据字段标题的显示方式，这里有左边(Left)与上方(Top)两种，请选择 Left，并单击“Next”按钮，如图 17.71 所示。

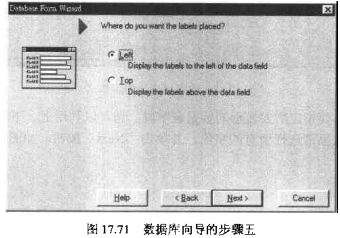


图 17.71　数据库向导的步骤五

步骤 7　接着，我们要决定这个即将要产生的 Form 是不是主要的 Form，及产生的方式。这里的产生方式是指是否将 DataSet 对象另外以 DataModule 存放。我们这里将这个 Form 设为主要的 Form，其产生方式为 Form Only，最后再单击“Finish”按钮，如图 17.72 所示。当您单击“Finish”按钮后，系统会自动产生相关的对象及 Form 对象，如图 17.73 所示。

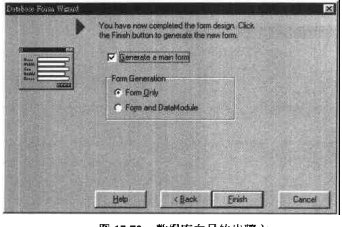


图 17.72　数据库向导的步骤六

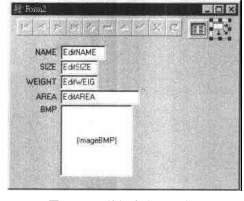


图 17.73　数据库向导的产生结果

步骤 8　保存工程，并按【F9】功能键运行工程。工程运行后，您可以利用 DBNavigator 对数据库进行各项操作，如图 17.74 所示。



图 17.74　数据库运行结果

17.4.2　向导创建 Master-Detail 的数据库程序

本小节将说明如何使用数据库向导创建 Master-Detail 的数据库程序。

范例 17-6

目的：本范例主要示范如何利用向导创建 Master-Detail 的数据库程序。

步骤 1　创建新的工程。

步骤 2　选择下拉式菜单“Database/Form Wizard”，打开“Database Form Wizard”对话框。

在“Form Options”中选择“Create a simple Form”，“DataSet Options”选择“Create a from using TTable Objects”。Form Option 主要是决定这个数据库应用程序的类型，若选择 Create a master/detail form，表示这个应用程序是由两个数据表所组成的，且这两个数据表成 master and detail 的关系。DataSet Options 用于决定所使用 DataSet 的种类，这里请选择 TQuery 对象。选择了适当的设置之后，请单击“Next”的按钮,如图 17.75 所示。

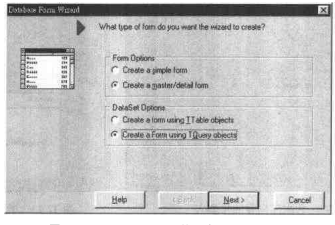


图 17.75 Master-Detail 数据库向导的步骤一

3 接着，要指定 Master 数据库表(Table)的位置，我们以 Delphi 6 所默认的数据表为例，这些数据表存在“C:\Program Files\Common Files\Borland Shared\Data”的目录下。我们选择 Parts.db，并单击“Next”按钮，如图 17.76 所示。

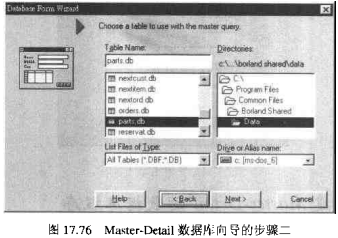


图 17.76 Master-Detail 数据库向导的步骤二

4 接着，我们要指定所要显示的 Master 数据表字段。您可以选利用上、下按键，指定字段出现的顺序，这里请选择所有的字段。并单击“Next”按钮，如图 17.77 所示。

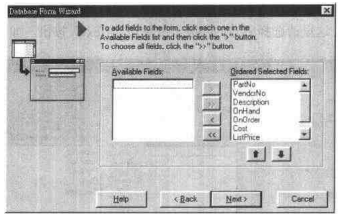


图 17.77　Master-Detail 数据库向导的步骤三

步骤 5　接着，我们要指定 Master 数据表字段的显示方式，这里的显示方式有水平(Horizontally)、垂直(Vertically)与表格(Grid)等显示方式。请选择水平的显示方式，并单击“Next”按钮，如图 17.78 所示。

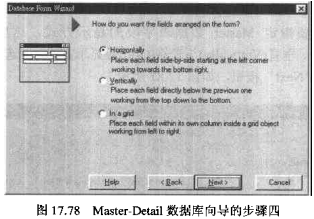


图 17.78　Master-Detail 数据库向导的步骤四

步骤 6　接着，要指定 Detail 数据库表(Table)的位置，我们以 Delphi 6 所默认的数据表为例，这些数据表存在“C:\Program Files\Common Files\Borland Shared\Data”的目录下。我们选择 Items.db，并单击“Next”按钮，如图 17.79 所示。

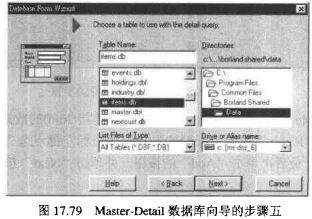


图 17.79　Master-Detail 数据库向导的步骤五

7 我们要指定所要显示的 Detail 数据表字段。您可以选利用上、下按键，指定字段出现的顺序，这里请选择所有的字段。并单击“Next”按钮，如图 17.80 所示。

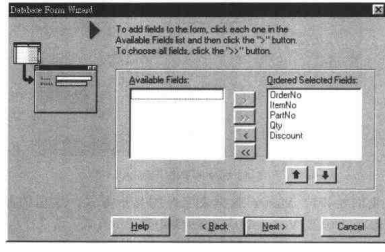


图 17.80 Master-Detail 数据库向导的步骤六

8 接着，我们要指定 Master 数据表字段的显示方式，这里的显示方式有水平(Horizontally)、垂直(Vertically)与表格(Grid)等显示方式。请选择表格(Grid)的显示方式，并单击“Next”按钮，如图 17.81 所示。

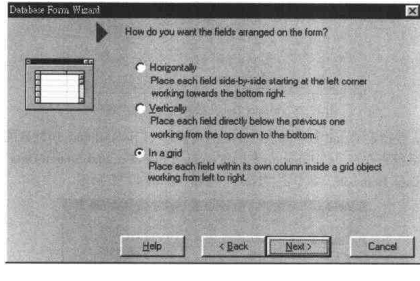


图 17.81 Master-Detail 数据库向导的步骤七

9 我们要指定 Master Table 与 Detail Table 之间连接(Join)的数据字段。请用鼠标左键点选“Detail Fields”中的“PartNO”，再用鼠标左键点选“Masster Fields”中的“PartNO”，最后再单击“Add”的按钮，并单击“Next”按钮，如图 17.82 所示。

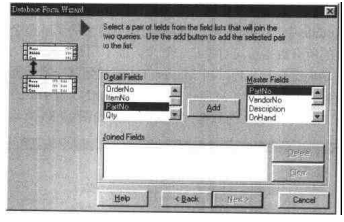


图 17.82　Master-Detail 数据库向导的步骤八

(10) 接下来，我们要决定这个即将要产生的 Form 是不是主要的 Form，及产生的方式。这里的产生方式是指是否要将 DataSet 对象另外以 DataModule 存放。我们这里将这个 Form 设为主要的 Form，其产生方式为“Form and DataModule”，最后再单击“Finish”按钮，如图 17.83 所示。当您单击“Finish”按钮后，系统会自动产生相关的对象及 Form 对象，如图 17.84 所示。

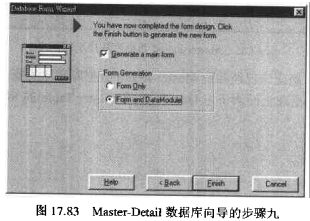


图 17.83　Master-Detail 数据库向导的步骤九

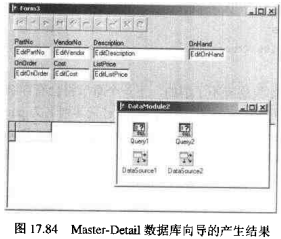


图 17.84　Master-Detail 数据库向导的产生结果

STEP 11 在图 17.84 中您会发现系统自动产生一个 TDataModule，并将相关的数据访问对象加到 DataModule 中，这样的作法可以使我们更容易的了解整个工程，再者我们可以将整个工程的所有数据访问对象集中管理，而不用重复使用。此时请先保存工程，并按【F9】功能键运行工程，当 Master Table 指到某一笔数据时，Detail 也会出现其对应的数据。例如我们将 Master Table 指到 PartNo 为 912 的数据时，Detail Table 会出现所有 PartNo 为 912 的数据，其运行结果如图 17.85 所示。

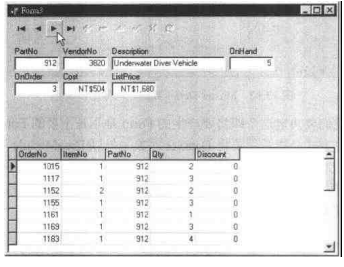


图 17.85 Master-Detail 数据库程序运行结果

通过上面的范例，您应该已经知道 Master-Detail 数据库程序的概念。我们现在就来看看其间的机制，我们可以点选上面例子中 Query1 对象(Master Table)的 SQL 属性，其内容如下：

```
Select
  parts."PartNo",
  parts."VendorNo",
  parts."Description",
  parts."OnHand",
  parts."OnOrder",
  parts."Cost",
  parts."ListPrice"
From "C:\Program Files\Common Files\Borland Shared\Data\parts.db"
As parts
```

关于这些 SQL 语法如果您有不了解的，下一个章节将会有详细的说明。上述 SQL 语法的目的，是由指定目录中获取 parts.db 的数据表并定义其别名为 parts，再引用这个别名获取 PartNo、VenderNo、Description、OnHand、onOrder、Cost、ListPrice 等字段的数据。我们可以用同样的方法获取 Query2 对象(Detail Table)的 SQL 属性，其内容如下：

```
Select
  items."OrderNo",
  items."ItemNo",
  items."PartNo",
```

```
  items."Qty",
  items."Discount"
From "C:\Program Files\Common Files\Borland Shared\Data\
        items.db"  As items
Where
  "C:\Program Files\Common Files\Borland Shared\Data\
      items.db"."PartNo" =:"PartNo"
```

在上面的 SQL 语法中有一个非常重要的 SQL 命令，那就是 Where，这个指定是用来设置 Master Table 与 Detail Table 之间的连接条件。

17.4.3　手动创建 *Master-Detail* 的数据库程序

Master-Detail 数据库程序在实作上还可以使用 TTable 对象，本小节将为大家说明如何使用 TTable 对象创建 Master-Detail 的数据库程序。



17-7

目的：本范例主要示范如何利用 TTable 对象创建 Master-Detail 的数据库程序。

STEP 1　创建新的工程。

STEP 2　在 TForm 对象中加入两个 TTable 对象、两个 T DataSource 及两个 TDBGrid 对象，其相关属性设置如表 17.8 所示。

表 17.8

对象	属性名称	属性设置值
TTable	Name	Table1
	Database	DBDEMOS
	TableName	PARTS.DB
	IndexFieldName	PartNo
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TTable	Name	Table2
	Database	DBDEMOS
	TableName	ITEMS.DB
	IndexFieldName	PartNo
	MasterSource	DataSource1
	MasterField	PartNo
	Active	True

续上表

对象	属性名称	属性设置值
TDataSource	Name	DataSource2
	DataSet	Table2
TDBGrid	Name	DBGrid2
	DataSource	DataSource2

③ 保存工程。

④ 按【F9】功能键运行工程。工程运行后，只要您移动 Master Table 的数据指针，则 Detail Table 就会出现所对应的数据，如图 17.86 所示。

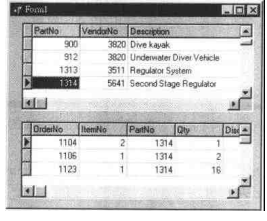


图 17.86 Master-Detail 数据库程序运行结果

17.5 数据库查询语言 SQL (Structured Query Language)

要写好数据库应用程序，除了要善用 Delphi 6 所提供的组件外，根据个人的经验，我们认为有一种“语言”您一定要知道，那就是数据库查询语言 SQL(Structured Query Language)。SQL 是一个非过程语言，用户通常不需要指定对数据的访问方法，而只要致力于要得到什么结果就可以，因为 SQL 是 RDBMS 的一部分，RDBMS 会自动指定最快速的访问方法，例如使用哪一个索引(Index)作为查询的依据等。再者 SQL 的语法类似于简单的英文，因此非常容易学习。以下就将各种常用的 SQL 加以说明，由于 SQL 命令可能会因为各种数据库的不同而有所不同。本小节所有的范例皆以 ORACLE 为主，因此如果有些 SQL 命令无法顺利运行，那么请查询数据库所提供的联机帮助。

17.5.1 SELECT 命令

SELECT 命令最主要的作用是从一个或多个表(Table)、视图(View)中快速的查询数据。SELECT 是 SQL 数据库查询语言中最常被使用的查询语言，其格式如下：

```
SELECT DISTINCT/ALL 字段名称 1,字段名称 2,...
```

```
FROM 数据表名称
WHERE 查询条件
GROUP BY 分组依据
HAVING 组内查询条件
ORDER BY 排序内容
```

以下就各个命令加以说明：

SELECT 与 FROM

在 SELECT 子句中有 DISTINCT 与 ALL 两个选项，其中 ALL 会列出所有的数据，而 DISTINCT 不会列出重复的数据。SELECT 与 FROM 是查询子句中必要的子句，RDBMS 会从 FROM 指定的数据表中列出 SELECT 所指定的字段。如果您不指定字段，可以使用“*”代表所有字段。您也可以使用汇总函数来找出符合条件的数据。常用的汇总函数如表 17.9 所示。

表 17.9

汇总函数名称与用法	说明
AVG(字段名称)	计算特定字段所有符合条件的平均值
COUNT(字段名称)	计算特定字段所有符合条件的条数(不含 NULL 值)
COUNT(*)	计算所有符合条件的数据条数(含 NULL 值与重复数据)
MAX(字段名称)	计算特定字段所有符合条件的最大值
MIN(字段名称)	计算特定字段所有符合条件的最小值
SUM(字段名称)	计算特定字段所有符合条件的总和

范例 17-8

目的：本范例主要示范 SELECT 与 FROM 等数据库查询语言。本范例会列出在 BDE 别名为 DBDEMOS 中的 employee 数据表中所有员工的人数及其平均工资。

1. 从“开始/程序/Borland Delphi 6”中点选“Database DeskTop”，以打开 Delphi 6 所附的数据库工具软件。
2. 选择下拉式菜单“File/New/SQL File”打开一个新的 SQL 文件(SQL Editor)。
3. 选择下拉式菜单“SQL/Select Alias...”，并选择 DBDEMOS 的 BDE 别名。
4. 在 SQL Editor 中输入 SQL 的命令，如下所示。

```
SELECT COUNT(*),AVG(SALARY) FROM EMPLOYEE
```

5. 保存 SQL File。
6. 选择下拉式菜单“SQL/Run SQL”以运行 SQL 命令，结果如图 17.87 所示。：

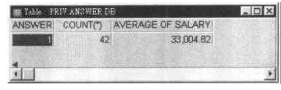


图 17.87

WHERE

WHERE 子句是用来指定查询的条件，在 WHERE 子句中可以配合各种运算符一起使用。常用的运算符如表 17.10 所示。

表 17.10

常用的运算符	说明
=	相等
<>或!=	不相等
>	大于
<	小于
>=	大于等于
<=	小于等于

WHERE 子句除了可以配合运算符使用外，更可以和下列的过滤条件一起使用，如表 17.11 所示。

表 17.11

常用的过滤条件	说明
BETWEEN A AND B	找出数据值介于 A 和 B 之间
LIKE	LIKE 是字符串部分相等的查询方式，通常配合%和_一起使用。%代表零个或多个字符，_代表一个字符。例如，LIKE K%就是要找出以 K 开头的字符串，LIKE K_就是要找出以 K 开头，而且只有两个字符的字符串
IS NULL	判断字段值是否为 NULL(空值)
IS NOT NULL	判断字段值是否不为 NULL(非空值)

WHERE 子句除此之外，也可以使用逻辑运算符来连接多个条件式。WHERE 字句常用的逻辑运算符如表 17.12 所示。

表 17.12

常用的逻辑运算符	说明
NOT	找出不符合指定条件的数据
AND	当两个指定条件都满足，此笔数据才算符合
OR	当两个指定条件式只要一个满足，此条数据就算符合

范例 17-9

目的：本范例主要示范 WHERE 子句中各种判断条件的使用方法。本范例会找出 LastName 为 B 开头，而且薪资在 40000 元以上的员工所有数据。操作过程请参考范例 13-8，以下仅列出 SQL 命令内容与运行结果。

SQL 命令内容：

```
SELECT * FROM EMPLOYEE WHERE   LASTNAME LIKE 'B%' AND SALARY
>= 40000
```

SQL 命令运行结果如图 17.88 所示。

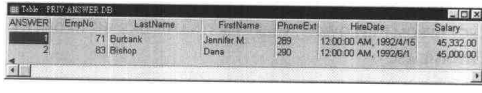


图 17.88

GROUP BY 与 HAVING

GROUP BY 子句可以指定字段作为分类依据，例如我们要求出各部门的工资总和就必须利用 GROUP BY 字句。GROUP BY 通常会配合 HAVING 子句一起使用，他们两者的关系就好像 SELECT 与 WHERE 一样。换言之，HAVING 就是 GROUP BY 的条件判断式。因此如果您没有使用 GROUP BY 子句，那您就不能使用 HAVING 子句。

范例 17-10

目的：本范例主要示范如何利用 GROUP BY 与 HAVING 子句，本范例要找出有一个人以上同时报到的到职日与报到人数。操作过程请看范例 17-8，以下仅列出 SQL 命令内容与运行结果。

SQL 命令内容：

```
SELECT HIREDATE,COUNT(*) FROM employee GROUP BY HIREDATE HAVING
COUNT(*)>1
```

SQL 命令运行结果如图 17.89 所示。

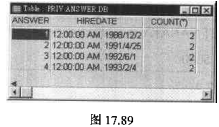


图 17.89

ORDER BY

ORDER BY 子句是用来决定查询结果的排列方式，ORDER BY 有 ASC 与 DESC 两种选择参数。其中 ASC(Ascend)代表数据由小到大排列，DESC(Descend)代表数据由大到小排列。



17-11

目的：本范例主要示范 ORDER BY 子句中各种判断条件的使用方法。本范例会找出 FirstName 为 B 开头的员工姓名数据，并以由小到大的顺序排列。操作过程请参考范例 17-8，以下仅列出 SQL 命令内容与运行结果。

SQL 命令内容：

```
SELECT * FROM EMPLOYEE   WHERE FIRSTNAME LIKE 'B%' ORDER BY
FIRSTNAME ASC
```

SQL 命令运行结果如图 17.90 所示。



ANSWER	EmpNo	LastName	FirstName	PhoneExt	HireDate	Salary
1	114	Parker	Bill	247	12:00:00 AM, 1993/6/1	35,000.00
2	4	Young	Bruce	233	12:00:00 AM, 1988/12/2	55,500.00

图 17.90

17.5.2 UPDATE 命令

UPDATE 命令是 SQL 数据库查询语言中最常被使用的数据更新命令，其格式如下：

```
UPDATE  数据表名称
SET  字段名称 1=表达式 1(或直接设置值),字段名称 2=表达式 2,...
WHERE  数据更新条件(请参考 17.5.1 小节)
```



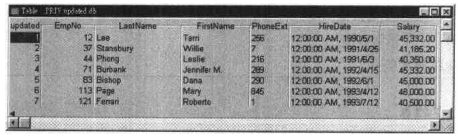
17-12

目的：本范例主要示范如何使用 UPDATE 命令。本范例会将工资在 40000 到 50000 之间的员工加薪 5%。操作过程请参考范例 13-7，以下仅列出 SQL 命令内容与运行结果。

SQL 命令内容：

```
UPDATE EMPLOYEE SET SALARY = SALARY * (1+ 0.05) WHERE SALARY
BETWEEN 40000 AND   50000
```

SQL 命令运行结果如图 17.91 所示。


Table - [EMPupdated.db]

updated	EmpNo	LastName	FirstName	PhoneExt	HireDate	Salary
1	12	Lee	Terri	255	12:00:00 AM, 1990/5/1	45,332.00
2	37	Stansbury	Willie	7	12:00:00 AM, 1991/4/25	41,185.20
3	44	Phong	Leslie	216	12:00:00 AM, 1991/6/3	40,350.00
4	71	Burbank	Jennifer M.	289	12:00:00 AM, 1992/4/15	45,332.00
5	83	Bishop	Dana	290	12:00:00 AM, 1992/6/1	45,000.00
6	113	Page	Mary	845	12:00:00 AM, 1993/4/12	48,000.00
7	121	Ferrari	Roberto	1	12:00:00 AM, 1993/7/12	40,500.00

图 17.91

17.5.3　DELETE 命令

DELETE 命令是 SQL 数据库查询语言中最常被使用的数据删除命令，其格式如下：

```
DELETE FROM 数据表名称
WHERE 数据删除条件(请参考 17.5.1 小节)
```



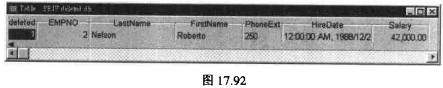
17-13

目的：本范例主要示范如何使用 DELETE 命令。本范例会删除员工编号为 2 的员工。操作过程请参考范例 17-8，以下仅列出 SQL 命令内容与运行结果。

SQL 命令内容：

```
DELETE FROM EMPLOYEE   WHERE EMPNO =   2
```

SQL 命令运行结果如图 17.92 所示。


Table - [EMPdeleted.db]

deleted	EMPNO	LastName	FirstName	PhoneExt	HireDate	Salary
1	2	Nelson	Roberto	250	12:00:00 AM, 1988/12/2	42,000.00

图 17.92

17.5.4　INSERT 命令

INSET 命令是 SQL 数据库查询语言中最常被使用的数据插入命令，其格式有两种，分述如下：

```
格式一、
INSET INTO 数据表名称(字段名称 1,字段名称 2,...)
VALUES (数值 1,数值 2,...)

格式二、
INSET INTO 数据表名称(字段名称 1,字段名称 2,...)
```

```
SELECT (数值 1,数值 2,...)
FROM    数据表名称
WHERE 过滤条件
```



17-14

目的：本范例主要示范如何使用 INSERT 命令。本范例会添加一条编号为 999 员工数据。操作过程请参考范例 17-8，以下仅列出 SQL 命令内容与运行结果。

SQL 命令内容：

```
INSERT INTO EMPLOYEE(EMPNO,LASTNAME) VALUES(999,'CARLOS')
```

SQL 命令运行结果如图 17.93 所示。

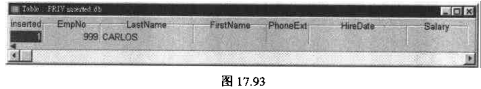


图 17.93

第 18 章

Delphi 6 数据库程序设计(二)

本章目标

本章将为大家介绍 Delphi 6 数据库程序设计常用的 TDataSet 对象，包括 TTable、TQuery 与 TStoredProc 等。这三个对象在 Delphi 6 数据库程序设计中占有相当大的份量，希望通过本章的范例说明可以让您对 Delphi 6 数据库程序设计有更清楚的概念。您可以在组件面板上的 BDE 页标签中找到本章所要介绍的所有组件。

本章主题

18.1 TdataSet 对象

18.2 Ttable 对象

18.3 Tquery 对象

18.4 TstoredProc 对象

在前一章(第 17 章)中曾为大家说明有关数据库的基本概念、Delphi 6 与各种数据库的连接设置、BDE Administrator 与 Database Desktop 等工具软件的使用时机与方法，以及使用向导创建简单或 Master-Detail 等类型的数据库程序，最后也为大家介绍常用的数据库查询语言 SQL(Structured Query Language)。相信您现在对数据库程序设计一点也不陌生才对，本章将更进一步地为大家说明 Delphi 6 提供的数据库对象。

18.1 TDataSet 对象

在 Delphi 6 所提供的数据库对象中以 TTable、TQuery 及 TStoredProc 等对象最为重要。因为这三个对象都是与数据库连接最主要的对象，这三个对象同样都继承自 TdataSet，如图 18.1 所示。

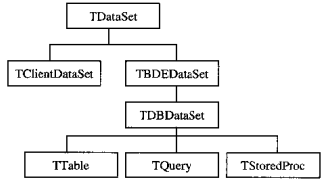


图 18.1 各种 DataSet 的继承关系

由图 18.1 可知 TTable、TQuery 与 TStoredProc 等数据库对象都是继承自 TDataSet 对象，以下就 TDataSet 的状态及各项常用的属性与操作方法加以说明。

18.1.1 TDataSet 对象状态

TDataSet 对象的状态概念会直接影响您使用 TTable、TQuery 与 TStoredProc 等对象。以下就 TDataSet 对象等可能的状态加以说明。TDataSet 对象共有 12 种可能的状态，如表 18.1 所示。

表 18.1

状态可能值	说明
dsInactive	表示目前 DataSet 是关闭的，因此它的数据是无法访问的
dsBrowse	表示目前 DataSet 的数据只可以浏览，因此无法改变其内容。当您第一次打开 DataSet，便处于这个状态
dsEdit	表示目前 DataSet 的数据是可以修改的
dsInsert	表示目前 DataSet 的数据是可以新增与插入的
dsSetKey	此状态只会出现于 TTable 对象，当您对 TTable 对象使用 SetRange 操作方法时，TTable 对象便处于这个状态

续上表

状态可能值	说明
dsCalcFields	当 DataSet 对象的 onCalcFields 事件(Event)被触发时，DataSet 对象就处于 dsCalcFields 状态
dsFilter	当DataSet对象的onFilterRecord事件(Event)被触发时，DataSet对象就处于dsFilter状态
dsNewValue	这是一个暂时性的状态，当 TField.NewValue 属性(property)被访问时，DataSet 对象就处于 dsNewValue 状态
DsOldValue	这是一个临时性的状态．当 TField.OldValue 属性(property)被访问时，DataSet 对象就处于 dsOldValue 状态
dsCurValue	这是一个临时性的状态，当 TField.CurValue 属性(property)被访问时，DataSet 对象就处于 dsCurValue 状态
dsBlockRead	当您调用 DataSet 的 Next 操作方法，但数据感知对象(Data-aware)的内容并没有被更新时，便处于 dsBlockRead 状态
dsInternalCalc	这是一个暂时性的状态，当 TField 正在进行内部计算时，DataSet 便处于 dsInternalCalc 状态

TDataSet 对象中有 dsInactive、dsBrowse、dsEdit、dsInsert 及 dsSetKey 等五种状态是应用程序中可以直接控制的，其关系如图 18.2 所示。

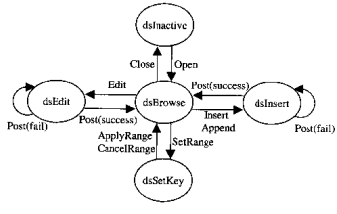


图 18.2　DataSet 的各种状态

18.1.2　*TDataSet* 对象属性

以下就 TDataSet 对象常用的属性加以说明：

Active 属性

Active 属性是用来判断或设置此 DataSet 对象是否为打开状态，当 Active 为 True 表示 DataSet 正处于打开的状态。

DatabaseName 属性

DatabaseName 属性是用来设置 DataSet 连接的数据库，它可以是一个 BDE 别名、TDataBase 对象名称，也可以是数据库所存在的目录。

TableName 属性

TableName 属性是 TTable 对象特有的属性，它是用来设置此 TTable 对象所连接数据库中的数据表名称(Table Name)。

IndexName 属性

IndexName 属性是 TTable 对象特有的属性，它是用来设置此 TTable 对象连接数据表所使用的索引名称(IndexName)。

IndexFieldNames 属性

IndexFieldNames 属性是 TTable 对象特有的属性，它是用来设置此 TTable 对象要以哪个字段(columns)来作为数据表索引(index)。

范例 18-1

目的：本范例主要说明如何使用指定路径打开数据表。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象、TLabel 对象、TEdit 对象及 TButton 对象，其相关属性设置如表 18.2 所示。

表 18.2

对象	属性名称	属性设置值
Ttable	Name	Table1
TdataSource	Name	DataSource1
TDBGrid	Name	DBGrid1
Tlabel	Name	Label1
	Capiton	数据库别名
Tedit	Name	Edit1
Tbutton	Name	Button1
	Caption	Open

STEP 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //设置 DataSource1 的 DataSet 属性
```

```
  DataSource1.DataSet:=Table1;

  //设置 DBGrid1 的 DataSource 属性
  DBGrid1.DataSource:=DataSource1;
  Table1.Active := False;
try
  //这里用一个不存在的 BDE 别名来打开 Table
  //如果您想正确的打开它，可以使用 DBDemos 的别名
  Table1.DatabaseName := 'CanNotOpenAlias';
  Table1.TableName:='parts.db';
  Table1.Active := True;
except
  //当使用 BDE 别名无法打开，可以直接指定数据表
  //所在的路径
  Table1.DatabaseName := 'c:\Program Files\Common Files\
                                        Borland Shared\data';
  Table1.TableName:='parts.db';
  Table1.Active := True;
end;
 //show 出 Table1 所使用的 DatabaseName
  Edit1.Text:=Table1.DatabaseName;
end;
```

④ 保存工程。

⑤ 按【F9】功能键运行工程。工程运行后，当您单击“Open”按钮，程序会尝试以“CanNotOpenAlias”的 BDE 别名打开指定的数据库。因为这个 BDE 别名我们并没有事先设置，所以它无法正确地被打开。此时会产生一个异常，我们在这个异常处理过程中指定数据表所在的路径，最后，程序会以这个指定的路径来打开它，如图 18.3 所示。



图 18.3　指定数据表路径打开数据表

BOF 属性

当 BOF 属性为 True，表示目前 DataSet 是处于第一条数据。

- *EOF 属性*

当 EOF 属性为 True，表示目前 DataSet 处于最后一条数据。

- *RecrodCount 属性*

RecordCount 属性用来表示目前 DataSet 所包含的所有数据条数。

- *RecordSize 属性*

RecordSize 属性表示此 DataSet 对象中每条数据的大小(Size)，其返回值是 Word 的类型。

- *FieldCount 属性*

FieldCount 属性表示此 DataSet 对象中的字段数。

- *Fields 属性*

Fields 属性继承自 TFields 的串行值，在 TFields 串行中是由多个 Tfield 对象所组成的。通过 TFields 属性可以获取某个字段值，此属性适用于不知道字段名称的情况。使用 Fields 属性所获取的值，其数据类型会根据您所指定的TField属性而决定，这些属性包括 AsBoolean、AsCurrency、AsDateTime、AsFloat、AsInteger、AsString 及 AsVariant 等。如果您想获取或设置某个字段值，其方法如下：

```
//获取某个字段值
Label1.Caption:=DataSet.Fields.Fields[第几个].AsString;
//设置某个字段值
DataSet.Fields.Fields[第几个].AsString:='指定字符串';
```

以上所示范的方法是通过 TDataSet 的 Fields 属性，也就是 TFields 对象的 Fields 属性来获取或设置字段值，事实上您可以使用比较简便的方法来获取或设置，其方法如下：

```
//获取某个字段值
Label1.Caption:=DataSet.Fields[第几个].AsString;
//设置某个字段值
DataSet.Fields[第几个].AsString:='指定字符串';
```

- *FieldValues 属性*

FieldValues 属性是获取或设置某个字段值的另一个方法，此属性适用于字段名称已知的情况。使用 FieldValues 属性所获取的值，其数据类型为 Variant 与 Fields 属性所获取的值有极大的不同。FieldValues 属性获取或设置某个字段值的方法如下：

```
//获取某个字段值
  //方法一
  Label1.Caption:= DataSet.FieldValues['字段名称'];
  //方法二
  Label1.Caption:= DataSet['字段名称'];
```

```
//设置某个字段值
  //方法一
  DataSet['字段名称']:='指定字符串'
  //方法二
  DataSet.FieldValues['字段名称']:='指定字符串'
```



除了可以使用 Fields 与 FieldValues 两个属性来获取或设置某个字段的数据外，您可以使用 FieldByName 的操作方法来完成，详细说明，请参考第 18.1.3 节。



18-2

目的：本范例主要说明如何获取有关 DataSet 对象的字段相关信息。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象及 TButton 对象，其相关属性设置如表 18.3 所示。

表 18.3

对象	属性名称	属性设置值
Ttable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	CUSTOMER.DB
	Active	True
TdataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
Tlabel	Name	Label1
Tbutton	Name	Button1
	Caption	获取字段相关信息

步骤 3 在 TButton 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I:Integer;
  Str:String;
begin
  with Table1 do begin
    Str:='这个数据表共有 '+IntToStr(RecordCount)+' 条'+ #10;
    Str:=Str+'每条数据的大小为: '+IntToStr(RecordSize)+' bytes'+
         #10;
```

```
    Str:=Str+'每条数据共有 '+IntToStr(FieldCount)+' 个字段'+
          #10;
    Str:=Str+'这些字段名称如下: '+ #10;
    for I := 0 to FieldCount - 1 do
       Str:=Str+'   '+Fields[I].FieldName+ #10;
  end;
  ShowMessage(Str);
end;
```

4 保存工程。

5 按【F9】功能键运行工程。工程运行后，此时 TTable 对象应该是处于打开状态，因为我们已经在步骤 2 将 TTable 对象的 Active 属性设为 True 了。如果您想获取有关这个 DataSet 的字段相关信息，请单击“获取字段相关信息”按钮即可，如图 18.4 所示。



图 18.4 获取 DataSet 字段相关信息

18.1.3 TDataSet 对象操作方法

以下就 TDataSet 对象常用的操作方法加以说明：

- *Open* 操作方法

Open 操作方法是用来打开 DataSet，其作用相当于将 Active 属性设为 True。

- *Close* 操作方法

Close 操作方法是用来关闭 DataSet，其作用相当于将 Active 属性设为 False。

- *Next* 操作方法

Next 操作方法会将 DataSet 的光标(Cursor)指到下一条，可利用 EOF 属性来判断是否已到了最后一条数据。

Prior 操作方法

Prior 操作方法会将 DataSet 的光标(Cursor)指到上一条，可利用 BOF 属性来判断是否已到了第一条数据。

Last 操作方法

Last 操作方法会将 DataSet 的光标(Cursor)指到最后一条。

First 操作方法

First 操作方法会将 DataSet 的光标(Cursor)指到第一条。

MoveBy 操作方法

MoveBy 操作方法会将 DataSet 的光标(Cursor)从目前的位置开始算起移到指定条数。例如：

```
//往下 2 条
Table1.MoveBy(2);

//往上 3 条
Table1.MoveBy(-3);
```

Insert 操作方法

Insert 操作方法会在 DataSet 光标(Cursor)目前的位置，以插入的方式新增一条空白的数据，如果您的数据表(Table)，有加入 not null 的限制条件，可能无法使用。但只要在 OnNewRecord 事件处理过程中加入插入数据字段的初值，那么就可以解决此限制条件。

Append 操作方法

Append 操作方法会在 DataSet 最后的位置，以附加的方式添加一条空白的数据，如果您的数据表(Table)，有加入 not null 的限制条件，可能无法使用。但只要在 OnNewRecord 事件处理过程中加入添加数据字段的初值，那么就可以解决此限制条件。

Edit 操作方法

Edit 操作方法将 DataSet 光标(Cursor)目前所在的数据记录(Record)设置为“可修改的状态(dsEdit)”，在这个状态，您可以修改目前这条数据，但必须使用 Post 操作方法来加以保存。如果您想放弃修改，可以调用 Cancel 操作方法。

Delete 操作方法

Delete 操作方法会将 DataSet 光标(Cursor)目前所在的数据记录(Record)加以删除。

Post 操作方法

如果您有调用 Edit、Insert 或 Append 等操作方法，必须使用 Post 操作方法保存所有的变

动。

Cancel 操作方法

如果您有调用 Edit、Insert 或 Append 等操作方法，可以使用 Cancle 操作方法来取消所有的变动。

AppendRecord 操作方法

AppendRecord 操作方法会在 DataSet 最后加入一条数据，这条数据值是直接被指定。其使用方法如下：

```
//假如您所指定的数据表(Table)只有三个字段，您可以直接指定
//值，也可以指定某个对象的属性(如 TEdit.Text)或以空值(null)替换。
DataSet.AppendRecord([‘一休和尚’,Null,SexEdit.Text])
```

InsertRecord 操作方法

InsertRecord 操作方法会在 DataSet 目前的位置加入一条数据，这条数据值直接被指定。其使用方法如下：

```
//假如您所指定的数据表(Table)只有三个字段，您可以直接指定
//值，也可以指定某个对象的属性(如 TEdit.Text)或以空值(null)取代。
DataSet.InsertRecord([‘一休和尚’,Null,SexEdit.Text])
```

FieldByName 操作方法

前面已经介绍了两种获取某个字段值的方法，那就是 DataSet 对象的 Fields 属性与 FieldValues 属性。这里要再为大家介绍一种获取某个字段值的方法。那就是 DataSet 对象的 FieldByName 操作方法，这个方法适用于字段名称已知的情况。在 FieldByName 操作方法之后可以直接指定数据类型，例如整数(AsInteger)、字符串(AsString)等(参考补充说明)，其方法如下：

```
//获取某个字段值
Label1.Caption:=DataSet.FieldByName('字段名称').AsString;

//设置某个字段值
DataSet.FieldByName('字段名称').AsString:=Edit1.Text;
```



前面曾经说过，我们可以在 FieldByName 操作方法之后直接指定数据类型，事实上这是因为 FieldByName 操作方法会以 TField 对象的数据类型返回您所指定的字段。因此 AsString、AsInteger 等特异功能要算是 TField 对象的属性(Properties)，表 18.4 就是这些数据类型转换的属性说明。

表 18.4

Tfield 对象属性	说明
AsBoolean	将指定字段的数据以布尔值(Boolean)类型表示
AsCurrency	将指定字段的数据以货币（Currency）类型表示

续上表

Tfield 对象属性	说明
AsDateTime	将指定字段的数据以日期时间(DataTime)类型表示
AsFloat	将指定字段的数据以浮点数(Float)类型表示
AsInteger	将指定字段的数据以整数(Integer)类型表示
AsString	将指定字段的数据以字符串(String)类型表示
AsVariant	将指定字段的数据以变体(Variant)类型表示

ClearFields 操作方法

ClearFields 操作方法会清除 DataSet 对象目前的数据记录(Record)。如果 DataSet 对象不是处于插入(dsInsert)或编辑(dsEdit)状态，而是调用 ClearFields 操作方法，会触发一个异常。其与 Delete 操作方法有相当大的不同。Delete 是删除目前的数据记录(Record)，而 ClearFields 是将数据记录清为空白。

IsEmpty 操作方法

IsEmpty 操作方法是用来判断 DataSet 对象是否为空的。如果 IsEmpty 返回值为 True，表示 DataSet 对象是空的(表示没有任何的数据记录 Record 在 DataSet 中)；反之，则不为空。

Refresh 操作方法

ReFresh 操作方法会放弃 DataSet 对象目前的所有数据记录(Record)，重新获取。

范例 18-3

目的：本范例主要说明如何使用 DataSet 对象的各种操作方法。

Step 1 创建新的工程。

Step 2 在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象、TGroupBox 对象、六个 TLabel 对象、六个 TEdit 对象及六个 TButton 对象，其相关属性设置如表 18.5 所示。

表 18.5

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	EMPLOYEE.DB
	IndexFieldName	EmpNo
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1

续上表

对象	属性名称	属性设置值
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TGroupBox	Name	GroupBox1
TLabel	Name	LabelNo
	Caption	员工编号
TLabel	Name	LabelExt
	Caption	分机号码
TLabel	Name	LabelLastName
	Caption	LastName
TLabel	Name	LabelFirstName
	Caption	FirstName
TLabel	Name	LabelHireDate
	Caption	到职日
TLabel	Name	LabelSalary
	Caption	工资
TButton	Name	ButtonFirst
	Caption	第一条
TButton	Name	ButtonNext
	Caption	下一条
TButton	Name	ButtonPrior
	Caption	上一条
TButton	Name	ButtonLast
	Caption	最后一条
TButton	Name	ButtonAppend
	Caption	插入
TButton	Name	ButtonModify
	Caption	修改
TButton	Name	ButtonDelete
	Caption	删除
TButton	Name	ButtonClear
	Caption	清除

③ 在 ButtonFirst 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonFirstClick(Sender: TObject);
begin
  //将 Cursor 移到第一条
```

```
    Table1.First;
  end;
```

④ 在 ButtonNext 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.ButtonNextClick(Sender: TObject);
  begin
    //将 Cursor 移到下一条
    Table1.Next;
  end;
```

⑤ 在 ButtonPrior 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.ButtonPriorClick(Sender: TObject);
  begin
    //将 Cursor 移到上一条
    Table1.Prior;
  end;
```

⑥ 在 ButtonLast 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.ButtonLastClick(Sender: TObject);
  begin
    //将 Cursor 移到最后一条
    Table1.Last;
  end;
```

⑦ 在 ButtonAppend 对象的 onClick 事件处理过程中加入以下代码：

```
  procedure TForm1.ButtonAppendClick(Sender: TObject);
  begin
    with Table1 do
    begin
      Try
        //进入新增模式(dsInsert)
        Append;

        //将指定数据写入各个字段
        FieldByName('EmpNo').AsInteger:=StrToInt(EditEmpNo.
                  Text);
        FieldByName('LastName').AsString:=EditLastName.Text;
        FieldByName('FirstName').AsString:=EditFirstName.Text;
        FieldByName('PhoneExt').AsString:=EditPhoneExt.Text;
        FieldByName('HireDate').AsDateTime:=StrToDateTime(Edit
                  HireDate.Text);
      FieldByName('Salary').AsFloat:=StrToFloat(EditSalary.Text);

        //将数据写回数据库，并回到浏览状态(dsBrowse)
        Post;
```

```
        //显示成功信息
        ShowMessage('数据已正确保存!!');
      Except
       //数据保存时发生错误，将 DataSet 设回浏览状态(dsBrowse)
        Cancel;
        ShowMessage('数据保存发生错误，请检查数据是否正确，
                              且没有重复的员工编号!!');
      end;
    end;
  end;
```

8 在 ButtonModify 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonModifyClick(Sender: TObject);
begin
  with Table1 do
  begin
    Try
      //进入修改模式(dsEdit)
      Edit;

      //将指定数据写入各个字段
      FieldByName('EmpNo').AsInteger:=StrToInt(EditEmpNo.
                Text);
      FieldByName('LastName').AsString:=EditLastName.Text;
      FieldByName('FirstName').AsString:=EditFirstName.Text;
      FieldByName('PhoneExt').AsString:=EditPhoneExt.Text;
      FieldByName('HireDate').AsDateTime:=StrToDateTime(
                EditHireDate.Text);
    FieldByName('Salary').AsFloat:=StrToFloat(EditSalary.Text);

      //将数据写回数据库，并回到浏览状态(dsBrowse)
      Post;
      //显示成功信息
      ShowMessage('数据已正确保存!!');
    except
     //数据保存时发生错误，将 DataSet 设回浏览状态(dsBrowse)
      Cancel;
      ShowMessage('数据保存发生错误，请检查数据是否正确，
                              且没有重复的员工编号!!');
    end;
  end;
end;
```

9 在 ButtonDelete 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
begin
  with Table1 do
  begin
    Try
      //删除数据
      Delete;
      //显示成功信息
      ShowMessage('数据已正确删除!!');
    except
     //数据保存时发生错误，将 DataSet 设回浏览状态(dsBrowse)
      ShowMessage('数据删除时发生错误!!');
    end;
  end;
end;
```

STEP 10 在 ButtonClear 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonClearClick(Sender: TObject);
begin
  //清除各个字段数据(不是数据库的内容)
  EditEmpNo.Text:='';
  EditLastName.Text:='';
  EditFirstName.Text:='';
  EditPhoneExt.Text:='';
  EditHireDate.Text:='';
  EditSalary.Text:='';
end;
```

STEP 11 在 TDBGrid 对象的 onCellClick 事件处理过程中加入以下代码：

```
procedure TForm1.DBGrid1CellClick(Column: TColumn);
begin
  //将 Table1 目前的数据记录(Record)存到 TEdit 对象中
  with Table1 do
  begin
    EditEmpNo.Text:=FieldByName('Empno').AsString;
    EditLastName.Text:=FieldByName('LastName').AsString;
    EditFirstName.Text:=FieldByName('FirstName').AsString;
    EditPhoneExt.Text:=FieldByName('PhoneExt').AsString;
    EditHireDate.Text:=FieldByName('HireDate').AsString;
    EditSalary.Text:=FieldByName('Salary').AsString;
  end;
end;
```

STEP 12 保存工程。

(13) 按【F9】功能键运行工程。这个工程主要是示范各种与 DataSet 相关的操作方法，如图 18.5 所示。



图 18.5 DataSet 对象综合说明

ControlsDisabled、DisableControls 与 EnableControls 操作方法

一般来说，DataSet 对象在使用上都会连接数据控制组件(Data Control)，例如 DBGrid、DBEdit、DBText 等。那么当我们移动 DataSet 的指针(Cursor)时，其所连接的数据控制组件也会同时显示指针最新所指的数据记录(Record)。这种设计有时候您会觉得很方便，但有时候，您可能会觉得非常的不理想。例如，您要重新设置 DataSet 对象的某个数据字段，此时您会发现数据控制组件上的数据不停的移动，不但有些难看，运行效率也会变差。因此您必须通过 DisableControls 来切断 DataSet 与数据控制组件(DataControl)之间的连接，如果您想恢复他们之间的连接，您可以使用 EnableControls 操作方法。通常我们在使用这两个操作方法前，我们会先通过 ControlDisabled 操作方法来获取目前 DataSet 与数据控制组件间的关系。

范例 18-4

目的：本范例主要说明如何在运行时期控制 DataSet 所连接的数据控制组件(Data Control)。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象及两个 TButton 对象，其相关属性设置如表 18.6 所示。

表 18.6 所示

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	EMPLOYEE.DB
	IndexFieldName	EmpNo
	Active	True

续上表

对象	属性名称	属性设置值
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TButton	Name	Button1
	Caption	切断连接，运行加薪
TButton	Name	Button2
	Caption	不切断连接，运行加薪

(3) 在 Button1 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Table1 do
  try
    //切断所连接的数据控制组件(Data Control)
    DisableControls;
    First;
    While NOT EOF  do
    begin
      Edit;
      //修改员工工资
      FieldByName('Salary').AsFloat:=
                    FieldByName('Salary').AsFloat * (1+0.05);
      Post;
      Next;
    end;
  finally
    //回到第一条数据，并恢复所连接的数据控制组件
    First;
    EnableControls;
  end;
end;
```

(4) 在 Button2 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  with Table1 do
  begin
    First;
    While NOT EOF  do
```

```
    begin
      Edit;
      //修改员工工资
      FieldByName('Salary').AsFloat:=
                    FieldByName('Salary').AsFloat * (1+0.05);
      Post;
      Next;
    end;
  end;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程。从这个工程的运行结果，相信您一定可以比较出这两个方法之间的差异，如图 18.6 所示。



图 18.6 运行时期控制数据控制组件

18.1.4 TDataSet 对象的事件处理

表 18.7 就是 DataSet 对象常用的事件处理说明。

表 18.7

DataSet 对象的事件处理	说明
BeforeCancel 事件	当系统运行 DataSet 调用 Cancel 操作方法之前会触发此事件，此事件通常在插入添加(Insert、Append)或编辑(Edit)时被触发
AfterCancel 事件	当系统运行 DataSet 调用 Cancel 操作方法之后会触发此事件
BeforeDelete 事件	当系统运行 DataSet 调用 Delete 操作方法之前会触发此事件
AfterDelete 事件	当系统运行 DataSet 调用 Delete 操作方法之后会触发此事件
OnDeleteError 事件	当系统运行 DataSet 调用 Delete 操作方法发生错误时会触发此事件。您有三种处理方式：放弃编辑但显示错误给用户(daFail)、放弃编辑但不显示错误给用户(daAbort)、重试(daRetry)。例如：您要放弃删除且响应用户错误信息，则在此事件处理过程中加入“Action:=daFail;”即可。如果您想获取错误信息名称，只要在事件处理过程中加入“ShowMessage(E.Message);”即可

续上表

DataSet 对象的事件处理	说明
BeforeEdit 事件	当系统运行 DataSet 调用 Edit 操作方法之前会触发此事件
AfterEdit 事件	当系统运行 DataSet 调用 Edit 操作方法之后会触发此事件
OnEditError 事件	当系统运行 DataSet 调用 Edit 操作方法发生错误时会触发此事件。当发生错误时，您有三种处理方式：放弃编辑但显示错误给用户(daFail)、放弃编辑但不显示错误给用户(daAbort)、重试(daRetry)。例如：您要放弃编辑且响应用户错误信息，则在此事件处理过程中加入“Action:=daFail;”即可。如果您想获取错误信息名称，只要在事件处理过程中加入“ShowMessage(E.Message);”即可
BeforeInsert 事件	当系统运行 DataSet 调用 Insert 操作方法之前会触发此事件
AfterInsert 事件	当系统运行 DataSet 调用 Insert 操作方法之后会触发此事件
BeforeOpen 事件	当系统运行 DataSet 调用 Open 操作方法之前会触发此事件
AfterOpen 事件	当系统运行 DataSet 调用 Open 操作方法之后会触发此事件
BeforePost 事件	当系统运行 DataSet 调用 Post 操作方法之前会触发此事件
AfterPost 事件	当系统运行 DataSet 调用 Post 操作方法之后会触发此事件
OnPostError 事件	当系统运行 DataSet 调用 Post 操作方法发生错误时会触发此事件。当发生错误时，您有三种处理方式：放弃编辑但显示错误给用户(daFail)、放弃编辑但不显示错误给用户(daAbort)、重试(daRetry)。例如：您要放弃保存且响应用户错误信息，则在此事件处理过程中加入“Action:=daFail;”即可。如果您想获取错误信息名称，只要在事件处理过程中加入“ShowMessage(E.Message);”即可
BeforeScroll 事件	当系统运行 DataSet 调用 First、Last、MoveBy、Next、Prior、FindKey、FindFirst、FindNext、FindLast、FindPrior 或 Locate 等操作方法之前会触发此事件
AfterScroll 事件	当系统运行 DataSet 调用 First、Last、MoveBy、Next、Prior、FindKey、FindFirst、FindNext、FindLast、FindPrior 或 Locate 等操作方法之后会触发此事件
OnCalcFields 事件	当系统重新计算 DataSet 的计算字段(calculated fields)时会触发此事件。详细说明请参考第 15 章
OnNewRecord 事件	当系统运行 DataSet 调用 Insert、Append 操作方法时会触发此事件，我们通常利用这个事件处理来设置由 Insert 与 Append 操作方法所添加数据记录(Record)的初值，以解决数据表 not null(不允许空值)的限制



18-5

目的：为了让用户了解 DataSet 对象的各种事件处理过程，本范例修改自范例 18-3。在范例 18-3 我们主要通过异常处理(Try...Except)来处理错误事件，本范例将通过 DataSet 对

象的事件处理来处理与控制错误事件的发生。

STEP 1 您可以将范例 18-3 加以复制一份，或重新创建；如果您是重新创建，请先完成范例 18-3 的步骤 2 与步骤 4 等工作。

STEP 2 在 TForm 对象中另外加入两个 TTable 对象，以显示触发的事件名称。其相关属性设置如表 18.8 所示。

表 18.8

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	触发的事件名称：
TLabel	Name	LabelEventName
	Caption	空白

STEP 3 重新改为“插入”TButton 对象的 onClick 事件处理过程，其更改后的代码如下：

```
procedure TForm1.ButtonAppendClick(Sender: TObject);
begin
  with Table1 do
  begin
    //进入插入模式(dsInsert)
    Append;

    //将指定数据写入各个字段
  FieldByName('EmpNo').AsInteger:=StrToInt(EditEmpNo.Text);
    FieldByName('LastName').AsString:=EditLastName.Text;
    FieldByName('FirstName').AsString:=EditFirstName.Text;
    FieldByName('PhoneExt').AsString:=EditPhoneExt.Text;
    FieldByName('HireDate').AsDateTime:=StrToDateTime(
              EditHireDate.Text);
    FieldByName('Salary').AsFloat:=StrToFloat(EditSalary.Text);

    //将数据写回数据库，并回到浏览状态(dsBrowse)
    Post;
    //显示成功信息
    ShowMessage('数据已正确保存!!');
  end;
end;
```

STEP 4 重新改为“修改”TButton 对象的 onClick 事件处理过程，其更改后的代码如下：

```
Procedure TForm1.ButtonModifyClick(Sender: TObject);
begin
  with Table1 do
  begin
    //进入修改模式(dsEdit)
```

```
    Edit;

    //将指定数据写入各个字段
  FieldByName('EmpNo').AsInteger:=StrToInt(EditEmpNo.Text);
    FieldByName('LastName').AsString:=EditLastName.Text;
    FieldByName('FirstName').AsString:=EditFirstName.Text;
    FieldByName('PhoneExt').AsString:=EditPhoneExt.Text;
    FieldByName('HireDate').AsDateTime:=StrToDateTime(
              EditHireDate.Text);
    FieldByName('Salary').AsFloat:=StrToFloat(EditSalary.Text);

    //将数据写回数据库，并回到浏览状态(dsBrowse)
    Post;
    //显示成功信息
    ShowMessage('数据已正确保存!!');
  end;
end;
```

STEP 5 重新改为“删除”等 TButton 对象的 onClick 事件处理过程，其更改后的代码如下：

```
procedure TForm1.ButtonDeleteClick(Sender: TObject);
begin
  with Table1 do
  begin
    //进入修改模式(dsEdit)
    Delete;
    //显示成功信息
    ShowMessage('数据已正确删除!!');
  end;
end;
```

STEP 6 在 TTable 对象的 AfterCancel 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
   //显示事件名称
   LabelEventName.Caption:='AfterCancel Event';
end;
```

STEP 7 在 TTable 对象的 AfterClose 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterClose(DataSet: TDataSet);
begin
   //显示事件名称
   LabelEventName.Caption:='AfterClose Event';
end;
```

STEP 8 在 TTable 对象的 AfterDelete 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='AfterDelete Event';
end;
```

Step 9 在 TTable 对象的 AfterEdit 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterEdit(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='AfterEdit Event';
end;
```

Step 10 在 TTable 对象的 AfterInsert 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterInsert(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='AfterInsert Event';
end;
```

Step 11 在 TTable 对象的 AfterOpen 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterOpen(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='AfterOpen Event';
end;
```

Step 12 在 TTable 对象的 AfterPost 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterPost(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='AfterPost Event';
end;
```

Step 13 在 TTable 对象的 AfterScroll 事件处理过程加入以下的代码：

```
procedure TForm1.Table1AfterScroll(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='AfterScroll Event';
end;
```

Step 14 在 TTable 对象的 BeforeCancel 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeCancel(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeCancel Event';
```

```
end;
```

Step 15 在 TTable 对象的 BeforeClose 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeClose(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeClose Event';
end;
```

Step 16 在 TTable 对象的 BeforeDelete 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeDelete(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeDelete Event';
end;
```

Step 17 在 TTable 对象的 BeforeEdit 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeEdit(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeEdit Event';
end;
```

Step 18 在 TTable 对象的 BeforeInsert 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeInsert(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeInsert Event';
end;
```

Step 19 在 TTable 对象的 BeforeOpen 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeOpen(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeOpen Event';
end;
```

Step 20 在 TTable 对象的 BeforePost 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforePost(DataSet: TDataSet);
begin
    //显示事件名称
    Label1EventName.Caption:='BeforePost Event';
end;
```

Step 21 在 TTable 对象的 BeforeScroll 事件处理过程加入以下的代码：

```
procedure TForm1.Table1BeforeScroll(DataSet: TDataSet);
```

```
begin
    //显示事件名称
    Label1EventName.Caption:='BeforeScroll Event';
end;
```

STEP 22 在 TTable 对象的 DeleteError 事件处理过程加入以下的代码：

```
procedure TForm1.Table1DeleteError(DataSet: TDataSet; E: EDatabaseError;
var Action: TDataAction);
begin
//使用 dbAbort 来处理 DeleteError，此时系统不会响应用用
//户错误信息
    Action:=daAbort;
    Label1.Caption:='发生删除错误:'+E.Message;
end;
```

STEP 23 在 TTable 对象的 EditError 事件处理过程加入以下的代码：

```
procedure TForm1.Table1EditError(DataSet: TDataSet; E: EDatabaseError; var
Action: TDataAction);
begin
//使用 dbFail 来处理 EditError，此时系统会响应用用户错
//误信息
    Action:=daFail;
end;
```

STEP 24 在 TTable 对象的 PostError 事件处理过程加入以下的代码：

```
procedure TForm1.Table1PostError(DataSet: TDataSet; E: EDatabaseError; var
Action: TDataAction);
begin
//使用 dbFail 来处理 PostError，此时系统会响应用用户错
//误信息
    Action:=daFail;
end;
```

STEP 25 保存工程。

STEP 26 按【F9】功能键运行工程。这个工程主要是示范 DataSet 各种相关的事件处理，如果您没有正确的指定数据类型(例如到职日)而保存的话，您将会触发 onEditError 事件，并且得到一个错误信息，如图 18.7 所示。



图 18.7　DataSet 对象各种事件综合说明

范例 18-6

目的：在范例 18-5 中，我们只是让您知道 DataSet 对象的事件处理时机与处理方法而已，本范例示范如何以 DataSet 对象的 AfterScroll 事件来控制数据的显示方法。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TTable 对象、一个 TDataSource 对象及一个 TDBGrid 对象，其相关属性设置如表 18.9 所示。

表 18.9

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	EMPLOYEE.DB
	IndexFieldName	EmpNo
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBNavigator	Name	DBNavigator
	DataSource	DataSource1
TDBEdit	Name	DBEdit1
	DataSource	DataSource1
	DataField	FirstName
TDBEdit	Name	DBEdit2
	DataSource	DataSource1
	DataField	HireDate

续上表

对象	属性名称	属性设置值
TDBEdit	Name	DBEdit3
	DataSource	DataSource1
	DataField	Salary

步骤 3 在 TTable 对象的 onAfterScroll 事件处理过程中加入以下的代码：

```
procedure TForm1.Table1AfterScroll(DataSet: TDataSet);
var
  I : Integer;
Begin
  if Dataset.FieldByName('Salary').AsInteger > 38000 then
     //将所有的 TDBEdit 组件的背景色变为黄色，并且设为只读
     For I := 0 to ComponentCount - 1 do
        begin
          if Components[i] is TDBEdit then
            begin
              (Components[i] as TDBEdit).Color := clYellow;
              (Components[i] as TDBEdit).ReadOnly := True;
            end;
        end
  else
     //将所有的 TDBEdit 组件的背景色变为白色，并且设为可读写
     For I := 0 to ComponentCount - 1 do
        begin
          if Components[i] is TDBEdit then
            begin
              (Components[i] as TDBEdit).Color := clWhite;
              (Components[i] as TDBEdit).ReadOnly := False;
            end;
        end;
end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程。工程运行后，您会发现只要工资大于 38,000 的员工数据都会以黄色的背景色来显示，而且数据是只读的。反之，其他员工数据是以白色的背景色来显示，而且数据是可以修改的，如图 18.8 所示。



图 18.8 DataSet 对象的 onAfterScroll 事件应用范例

18.2　TTable 对象

TTable 对象是继承自 TDataSet 对象，因此前一小节所说明的大部分“属性”、“操作方法”与“事件处理过程”都可以使用，所以不再赘述。以下仅就“数据过滤”与“数据搜索”两个主题加以说明。

18.2.1　TTable 对象数据过滤

如果您要通过 TTable 对象来过滤数据，您可以使用 Filter 属性或通过 onFilterRecord 事件处理来完成。这两种方法的运行结果是“且(AND)”，也就是说，每一条数据必须同时符合 Filter 属性或 onFilterRecord 事件所设置的过滤条。

18.2.1.1　利用 Filter 属性进行数据过滤

Filter 属性是一种很简单的数据过滤方法，您只要在 Table 对象的 Filter 属性输入“过滤条件”，再将 Filtered 属性设为 True 即可，您可以在设计时期或运行时期来设置，也可以利用 FilterOptions 属性来决定过滤的条件。FilterOptions 属性有两种值，分别说明如下：

- *oCaseInsensitive 属性值*

foCaseInsensitive 表示如果过滤字段为字符串类型，那么过滤条件是不分字母大小写的，也就是 Case Insensitive。FilterOption 属性中，如果没有包含 foCaseInsensitive，则代表过滤条件是有大小写区分的。

- *foNoPartialCompare 属性值*

foNoPartialCompare 表示会将星号(*)作为字符处理而非通用字符。FilterOption 属性中，如果没有包含 foNoPartialCompare，则代表星号(*)为通用字符。

范例 18-7

目的： 本范例主要说明如何通过 Filter 属性来过滤数据。

STEP 1　创建新的 Project。

STEP 2　在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象及 TButton 对象，其相关属性设置如表 18.10 所示。

表 18.10

对象	属性名称	属性设置值
Ttable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	Employee.db
	Active	True

续上表

对象	属性名称	属性设置值
TdataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
Tbutton	Name	Button1
	Caption	数据过滤

3 在 TButton 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str:String;
  Flag:Boolean;
begin
  //先通过 InputQuery 对象来输入过滤条件
  Str:='';
  Flag:=InputQuery('请入过滤条件','过滤条件',Str);
  if Flag then
  begin
    //进行过滤
    Try
      Table1.Filtered:=False;
      Table1.Filter:=Str;
      Table1.Filtered:=True;
    //过滤条件错误
    except
      ShowMessage('过滤条件错误，无法运行');
      Table1.Filtered:=False;
    end;
  end;
end;
```

4 保存工程。

5 按【F9】功能键运行工程。工程运行后，您可以单击“数据过滤”按钮，并输入过滤条件，此时 TDBGrid 便会实时显示过滤结果，如图 18.9 所示。

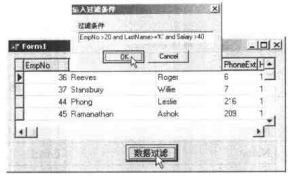


图 18.9　Table 对象的 Filter 属性

18.2.1.2　利用 *onFilterRecord* 事件进行数据过滤

Filter 属性通常是用来处理简单的过滤条件，如果您有比较复杂的过滤条件，那您可以考虑通过 onFilterRecord 事件处理过程来完成。这里请大家注意一件事，那就是 onFilterRecord 的运行效率不是那么的好，因此笔者并不是那么建议使用。事实上，笔者比较建议您使用 TQuery 对象或 TStoredProc，因为他们有比较好的运行效率，而且对于将来您在学习其他程序语言或数据库(DBMS)都会有相当的帮助。OnFilterRecord 事件处理过程中有 DataSet 与 Accept 两个参数，前者代表 TDataSet，也就是指 TTable、TQuery 或 TstoredProc 等 DataSet 对象。而 Accept 参数代表此条记录是否符合过滤的条件，当 Accept 为 True，则表示符合过滤条件；反之，则不符合。OnFilterRecord 事件处理过程的声明如下：

```
procedure Table1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
```

范例 18-8

目的：本范例主要说明如何通过 onFilterRecord 事件处理过程来过滤数据。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象及 TButton 对象，其相关属性设置如表 18.11 所示。

表 18.11

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	Employee.db
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1

续上表

对象	属性名称	属性设置值
TButton	Name	Button1
	Caption	数据过滤
TEdit	Name	Edit1
TEdit	Name	Edit2
TLabel	Name	Label1
	Caption	员工编号
TLabel	Name	Label2
	Caption	至

STEP 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //切换过滤条件
  Table1.Filtered := NOT Table1.Filtered;
end;
```

STEP 4 在 TTable 对象的 onFilterRecord 事件处理过程加入以下代码：

```
procedure TForm1.Table1FilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
begin
  //转换过滤条件，并决定此时数据是否符合资格
  if (Table1.FieldByName('EmpNo').AsInteger >
                                    StrToInt(Edit1.Text) ) and
    (Table1.FieldByName('EmpNo'). AsInteger <
                                    StrToInt(Edit2.Text)) then
    Accept:=True
  else
    Accept:=False;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程。工程运行后，您可以单击“数据过滤”按钮，并输入过滤条件，此时 TDBGrid 便会实时显示过滤结果，如图 18.10 所示。



图 18.10 Table 对象的 onFilterRecord 操作方法

18.2.2　TTable 对象数据搜索

TTable 对象提供了 FindKey、Locate 及 LookUp 等数据搜索的方法，基于效率上的考虑，我们在此仅对 Locate 与 LookUp 等方法加以说明。

18.2.2.1　Locate 搜索

Locate 是 TTable 对象常用的数据搜索方法之一。您只要指定所要搜索的字段名称、搜索值与搜索方式即可。当数据搜索成功会返回 True，并将数据指针指向该条数据；反之，则返回 False，并将数据指针停在原来的位置。其声明原型如下：

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions): Boolean; virtual;
```

从 Locate 声明中我们可以了解“数据搜索方式”是继承自 TLocateOptions，而 TLocateOptions 有 loCaseInsensitive 与 loPartialKey 两种选择，前者代表搜索时字母不分大小写，后者代表搜索时只要部分数据符合，此条数据就算符合。

范例 18-9

目的：本范例主要说明如何通过 Locate 函数来搜索数据。

STEP 1　创建新的一程。

STEP 2　在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象及 TButton 对象，其相关属性设置如表 18.9 所示。

表 18.9

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	Employee.db
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TButton	Name	Button1
	Caption	单一字段搜索
TButton	Name	Button2
	Caption	多个字段搜索
TEdit	Name	Edit1

续上表

对象	属性名称	属性设置值
TEdit	Name	Edit1
TLabel	Name	Label1
	Caption	员工编号
TLabel	Name	Label2
	Caption	分机号码

步骤 3 分别在 Button1 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //单一字段数据搜索
  if Table1.Locate('EmpNo',Edit1.Text,[loCaseInsensitive])
  then    ShowMessage('找到您所指定的数据')
  else    showMessage('找不到您所指定的数据');
end;
```

步骤 4 分别在 Button2 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //多个字段搜索数据(uses Variants)
  if Table1.Locate('EmpNo;PhoneExt',VarArrayOf(
                [Edit1.Text,Edit2.Text]),[loCaseInsensitive]) then
     ShowMessage('找到您所指定的数据')
  else
     showMessage('找不到您所指定的数据');
end;
```

步骤 5 保存工程。

步骤 6 按【F9】功能键运行工程。工程运行后，您可以单击“数据搜索”按钮，当数据成功的搜索，此时数据指针(Cursor)便会指到搜索的记录上。否则，就会显示“找不到指定数据”的信息，如图 18.11 所示。



图 18.11 Table 对象的 Locate 函数

18.2.2.2　LookUp 搜索

LookUp 是 TTable 对象常用的数据搜索方法之一。您只要指定所要搜索的字段名称、搜索值及所要返回的字段值即可。当数据搜索成功会以变量的数据类型返回所指定的字段值。反之，则返回 null(空值)。其与 Locate 最大的不同在于 LookUp 并不会改变目前的数据指针。其声明原型如下：

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; virtual;
```

范例 18-10

目的：本范例主要说明如何通过 LookUp 函数来搜索数据。

STEP 1　创建新的工程。

STEP 2　在 TForm 对象中加入一个 TTable 对象、TDataSource 对象、TDBGrid 对象及 TButton 对象，其相关属性设置如表 11.13 所示。

表 11.13

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	Employee.db
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TLabel	Name	LabelNo
	Caption	员工编号
TLabel	Name	LabelName
	Caption	LastName
TEdit	Name	EditNo
TEdit	Name	EditName
TButton	Name	Button1
	Caption	返回单一字段
TButton	Name	Button2
	Caption	返回多字段

STEP 3　分别在 Button1 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
    FindResult:Variant;
    Str:String;
begin
   //搜索数据(搜索多字段，返回单一字段)  uses Variants
   FindResult:=Table1.Lookup('EmpNo;LastName',VarArrayOf(
                    [EditNo.Text,EditName.Text]),'Salary');
   if  NOT VarIsNull(FindResult) then
       begin
         Str:='您所搜索的数据如下:'+ #10;
         Str:=Str+varToStr(FindResult)+ #10;
         showMessage(Str);
       end
   else
       ShowMessage('找不到您所指定的数据');
end;
```

步骤 4 分别在 Button2 对象的 onClick 事件处理过程加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
var
    FindResult:Variant;
    Str:String;
begin
   //搜索数据  uses Variants
   //情况一、搜索单一字段，返回多字段
   //FindResult:=Table1.Lookup('EmpNo',EditNo.Text,'LastName;
FirstName;Salary');
   //情况二、搜索多字段，返回多字段
   FindResult:=Table1.Lookup('EmpNo;LastName',VarArrayOf(
                    [EditNo.Text,EditName.Text]),'LastName;
                    FirstName; Salary');
   if  NOT VarIsNull(FindResult) then
       begin
         Str:='您所搜索的数据如下:'+ #10;
         Str:=Str+varToStr(FindResult[0])+ #10;
         Str:=Str+varToStr(FindResult[1])+ #10;
         Str:=Str+varToStr(FindResult[2])+ #10;
         showMessage(Str);
       end
   else
       ShowMessage('找不到您所指定的数据');
end;
```

5 保存工程。

6 按【F9】功能键运行工程。工程运行后，您可以单击“搜索数据”按钮，当数据成功的搜索，此时会显示搜索的返回值。否则，就会显示“找不到指定数据”的信息，如图 18.12 所示。

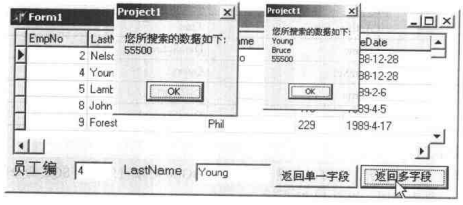


图 18.12　Table 对象的 Lookup 函数

18.3　TQuery 对象

TQuery 对象继承自 TDataSet 对象，因此章节 18.1 所说明大部分的“属性”、“操作方法”与“事件处理过程”都可以使用。因此不再赘述。以下仅就“设计时期的 TQuery 对象”与“运行时期的 TQuery 对象”两个主题加以说明。

18.3.1　设计时期的 TQuery 对象

TQuery 对象与其他 DataSet 对象最大的不同就是 TQuery 对象可以直接使用 SQL(数据库查询语言,Structured Query Language)，使用 SQL 的好处我们也一再的强调与说明，这里不再赘述。TQuery 对象有一个很重要的属性，那就是 SQL 属性，它继承自 TString 对象，主要是用来保存 SQL 命令。如果您输入适当的 SQL 指定，再将 Active 属性设为 True 即可运行指定的 SQL 命令。事实上它与 TTable 对象在使用上并没有太大的差异。

范例 18-11

目的：本范例主要说明设计时期设置 SQL 属性的 TQuery 对象。

1 创建新的工程。

2 在 TForm 对象中加入一个 TQuery 对象、TDataSource 对象、TDBGrid 对象及 TButton 对象，其相关属性设置如表 18.14 所示。

表 18.14

对象	属性名称	属性设置值
TQuery	Name	Query1
	DatabaseName	DBDEMOS
	SQL	Select * from Employee.db
	Active	True
TDataSource	Name	DataSource1
	DataSet	Query1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1

3 保存工程。

4 通过上面的设置，您可以在设计时期从 DBGrid 对象看到符合 SQL 命令的结果。运行时期也相同，如图 18.13 所示。

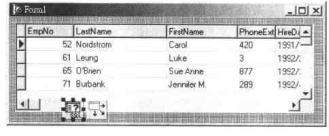


图 18.13 设计时期的 TQuery 对象

TQuery 对象除了 SQL 属性外，另有一个常用的属性，那就是 TParams。TParans 主要是用来设置 SQL 命令所使用的变量类型与初值。如果您想传参数给 SQL 属性，您只要在 SQL 的变量名称之前加入一个“:”符号即可。如下所示：

```
Select * from employee.db      where empno > :v_EmoNo
```

此时，请您选择 TQuery 对象的 Params 属性，就可以设置参数的类型或初值，如图 18.14 所示。

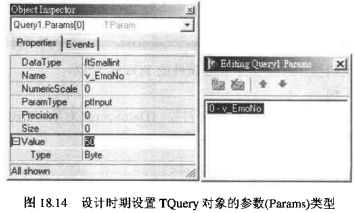


图 18.14 设计时期设置 TQuery 对象的参数(Params)类型

在图 18.14 中，我们主要示范如何设置 TQuery 对象的参数。这里我们将此 v_EmoNo 参数的数据类型(DataType)设置成整数(ftSmallint)，参数类型(ParamType)设为输入类型(ptInput)。最后再将参数的初值设为 50。当您设置完毕，重新再将 TQuery 对象的 Active 属性设置 True 时，此时 TDBGrid 对象便会显示员工编号大于 50 的数据。

18.3.2　运行时期的 TQuery 对象

TQuery 对象除了可以在设计时期设置相关的 SQL 与 Params 属性外，也可以在运行时期动态输入 SQL 命令与设置参数，我们就以范例说明。



18-12

目的：本范例主要说明运行时期设置 SQL 属性的 TQuery 对象。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TTQuery 对象、TDataSource 对象、TDBGrid 对象、TButton 对象、两个 TLabel 对象及两个 TEdit 对象，其相关属性设置如表 18.15 所示。

表 18.15

对象	属性名称	属性设置值
TQuery	Name	Query1
TDataSource	Name	DataSource1
TDBGrid	Name	DBGrid1
TButton	Name	Button1
	Caption	查询
TLabel	Name	Label1
	Caption	员工编号
TLabel	Name	Label2
	Caption	至
TEdit	Name	Edit1
TEdit	Name	Edit2

STEP 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  DataSource1.DataSet:=Query1;
  DBGrid1.DataSource:=DataSource1;
  with Query1 do
  begin
    DatabaseName:='DBDEMOS';
    Close;
    SQL.Clear;
```

```
    {加入 SQL 命令}
    SQL.Add('Select * from employee.db where empno
                    > :v_empno1 and empno < :v_empno2');
    {定义参数}
    Params[0].DataType := ftSmallint;
    Params[0].ParamType:= ptInput;
    Params[0].value:=Edit1.Text;
    Params[1].DataType := ftSmallint;
    Params[1].ParamType:= ptInput;
    Params[1].value:=Edit2.Text;
    {通过 BDE 作最优化的处理}
    Prepare;
    {也可以使用 ExecSQL,但 ExecSQL 通常使用于 INSERT,
      UPDATE,DELETE,CREATE TABLE 等, 至于 SELECT
      我们建议以 Open 代替 ExecSQL}
    Open;
  end;
end;
```

4. 保存工程。

5. 按【F9】功能键运行工程。工程运行后您可以输入指定范围的员工编号，再单击“查询”按钮，系统此时会动态地加入 SQL 命令及设置参数，并将查询结果加以显示，如图 18.15 所示。



图 18.15　运行时期的 TQuery 对象

18.3.3　动态产生 SQL 语句

在实务上，我们可能无法事先写好 SQL 语句，例如我们提供多个查询条件，但用户可能仅填写其中少数几个字段。因此我们必须动态产生 SQL 语句，以满足用户需求。

范例 18-13

目的：本范例主要说明如何动态产生 SQL 命令。

step 1 创建新的工程。

step 2 在 TForm 对象中加入一个 TQuery 对象、一个 TdataSource 对象、一个 TDBGrid 对象、一个 TButton 对象、两个 TLabel 对象及四个 TEdit 对象，其相关属性设置如表 18.16 所示。

表 18.16

对象	属性名称	属性设置值
TQuery	Name	Query1
TDataSource	Name	DataSource1
TDBGrid	Name	DBGrid1
TButton	Name	Button1
	Caption	查询
TLabel	Name	Label1
	Caption	员工编号
TLabel	Name	Label2
	Caption	LastName
TEdit	Name	EditEmpNoStart
TEdit	Name	EditEmpNoEnd
TEdit	Name	EditNameStart
TEdit	Name	EditNameEnd

step 3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str : String;
  Flag : Integer;
begin
  //连接对象
  DataSource1.DataSet:=Query1;
  DBGrid1.DataSource:=DataSource1;

  //设置变量初值
  Flag :- 0;
  with Query1 do
  begin
    DatabaseName:='DBDEMOS';
    //先关闭
    Close;

    //清除先前的 SQL
    SQL.Clear;
```

```
        //组合 SQL 语句
        Str := 'Select * from employee.db ';
        //判断 EmpNo 的条件
        if EditEmpNoStart.Text <> '' then
          begin
            Str := Str + 'Where empno between ' +
                  EditEmpnoStart.Text + ' and ' +
                  EditEmpnoEnd.Text;
            Flag := 1;
          end;
        //判断 Last Name 的条件
        if EditNameStart.Text <> '' then
          begin
            if Flag = 0 then Str := Str + ' where '
            else Str := Str + ' and ';
            Str := Str + 'LastName between ''' +
                  EditNameStart.Text + ''' and ''' +
                  EditNameEnd.Text + '''';
          end;
        //加入 SQL 语句
        SQL.Add(Str);
        //通过 BDE 作最优化的处理
        Prepare;
        //打开
        Open;
      End;
  end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程。工程运行后，您可以输入指定的条件进行查询，如图 18.16 所示。



图 18.16 动态产生 SQL 命令范例

18.3.4　利用 SQL 语句动态查询(以 Oracle 为例)

您除了可以在程序中利用技巧将用户所要查询的数据转换为 SQL 语句外，我们也可以利用 SQL 的命令来完成它。本小节将以 Oracle 为例说明如何通过 SQL 命令的特性来完成动态查询。至于 Microsoft SQL Server 的用法请参考第 18.3.5 节。本小节将使用 DECODE 的命令来完成动态 SQL 语句查询，有关 DECODE 的用法如下：

```
DECODE (测试值,
判断式一, 返回值一
[[,判断式二, 返回值二] [...]]
[,其他条件的返回值]
)
```

范例 18-14

目的：本范例主要说明如何利用 SQL 命令达到动态查询的目的(以 Oracle 为例)。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象上加入一个 TDatabase 对象、一个 TQuery 对象、一个 TDataSource 对象、一个 TDBGrid 对象、一个 TButton 对象、四个 TLabel 对象及四个 TEdit 对象，其相关属性设置如表 18.17 所示。

表 18.17

对象	属性名称	属性设置值
TDatabase	Aliasname	MyOracle
	Name	Database1
	DatabaseName	Abc
	HandleShared	True
	LoginPrompt	False
	Params	UserName=scott Password=tiger
	Connected	True
TQuery	Name	Query1
	DatabaseName	Abc
	SQL	(详述在后)
	Params	(详述在后)
TDataSource	Name	DataSource1
	DataSet	Query1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TButton	Name	Button1
	Caption	查询

续上表

对象	属性名称	属性设置值
TEdit	Name	EditEmpNoStart
TEdit	Name	EditEmpNoEnd
TEdit	Name	EditENameStart
TEdit	Name	EditENameEnd
TLabel	Name	Label1
	Caption	EmpNo
TLabel	Name	Label2
	Caption	至
TLabel	Name	Label3
	Caption	EName
TLabel	Name	Label4
	Caption	至

步骤 3 在 TQuery 对象的 SQL 属性中填入以下的代码：

```
Select *
from  emp
where (
      decode(:StartEmpno,null,0,:StartEmpno) <=
      decode(:StartEmpNo,null,0,Empno)
      and
      decode(:EndEmpNo,null,0,:EndEmpNo) >=
      decode(:EndEmpNo,null,0,Empno)
      )
      and
      (
      decode(:StartEName,null,'x',:StartEName) <=
      decode(:StartEName,null,'x',substr(EName,1,1))
      and
      decode(:EndEName,null,'x',:EndEName) >=
      decode(:EndEName,null,'x',substr(EName,1,1))
      )
order by empno
```

步骤 4 为 TQuery 对象 Params 属性中的每一个参数指定 DataType、ParamType 及 Value.Type。由于每一个参数均出现多次，因此您只要设置其中一个参数即可。相关参数设置如表 18.18 所示。

表 18.18

参数名称	属性名称	属性设置值
StartEmpno	DataType	ftInteger
	ParamType	ptInput
	Value.Type	Byte
EndEmpno	DataType	ftInteger
	ParamType	ptInput
	Value.Type	Byte
StartEName	DataType	ftString
	ParamType	ptInput
	Value.Type	String
EndEName	DataType	ftString
	ParamType	ptInput
	Value.Type	String

5 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Query1 do
  begin
    //先关闭
    Close;
    //更改参数值
    if trim(EditEmpNoStart.text) = '' then
      Params[0].value := null
    else
      Params[0].value := StrToInt(EditEmpNoStart.text);
    //更改 JobID2 的参数值
    if trim(EditEmpNoEnd.text) = '' then
      Params[3].value := null
    else
      Params[3].value := StrToInt(EditEmpNoEnd.text);
    //更改参数值
    Params[6].value := EditNameStart.text;
    Params[9].value := EditNameEnd.text;
    {通过 BDE 作最优化的处理}
    Prepare;
    //打开
    Open;
```

```
    end;
end;
```

6 保存工程。

7 按【F9】功能键运行工程。工程运行后，您可以任意输入查询条件。下图以查询员工编号(EmpNo)介于 7400~7800 之间，而且员工姓名(EName)是介于字母 A~D 之间的员工数据为例，如图 18.17 所示。

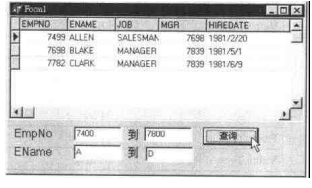


图 18.17 利用 SQL 命令动态查询(以 Oracle 为例)

18.3.5 利用 SQL 语句动态查询(以 MSSQL 为例)

您除了可以在程序中利用技巧将用户所要查询的数据转换为 SQL 语句外，我们也可以利用 SQL 的命令来完成它。本小节将以 Microsoft SQL Server 为例，说明如何通过 SQL 命令的特性来完成动态查询。至于 Oracle 的用法请参考第 18.3.4 节。本小节将使用 CASE 的命令来完成动态 SQL 语句查询，有关 CASE 的用法如下：

```
CASE 测试值
WHEN 判断式一 THEN 返回值一
[[WHEN 判断式二 THEN 返回值二] [...]]
[ELSE 其他条件的返回值]
END
```



18-15

目的：本范例主要说明如何利用 SQL 命令达到动态查询的目的(以 MS SQL 为例)。

1 创建新的工程。

2 在 TForm 对象上加入一个 TDatabase 对象、一个 TQuery 对象、一个 TDataSource 对象、一个 TDBGrid 对象、一个 TButton 对象、四个 TLabel 对象及四个 TEdit 对象，其相关属性设置如表 18.19 所示。

表 18.19

对象	属性名称	属性设置值
TDatabase	Aliasname	MyMSSQL
	Name	Database1
	DatabaseName	Abc
	HandleShared	True
	LoginPrompt	False
	Params	UserName=sa Password=
	Connected	True
TQuery	Name	Query1
	DatabaseName	Abc
	SQL	(详述在后)
	Params	(详述在后)
TDataSource	Name	DataSource1
	DataSet	Query1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TButton	Name	Button1
	Caption	查询
TEdit	Name	EditJobId1
TEdit	Name	EditJobId2
TEdit	Name	EditName1
TEdit	Name	EditName2
TLabel	Name	Label1
	Caption	Job ID
TLabel	Name	Label2
	Caption	至
TLabel	Name	Label3
	Caption	First Name
TLabel	Name	Label4
	Caption	至

STEP 3 在 TQuery 对象的 SQL 属性中填入以下的代码：

```
Select Job_Id,Fname,LName
from  employee
where (
          ((case :Job_id1 when null then 0 else job_id end) >=
```

```
    (case :Job_id1 when null then 0 else :job_id1 end)) and
   ((case :Job_id2 when null then 0 else job_id end) <=
    (case :Job_id2 when null then 0 else :job_id2 end))
     ) and (
   ((case :fname1 when null then 'x' else fname end) >=
    (case :fname1 when null then 'x' else :fname1 end)) and
  ((case :fname2 when null then 'x' else fname end) <=
   (case :fname2 when null then 'x' else :fname2 end)))
order by job_id
```

Step 4 为 TQuery 对象 Params 属性中的每一个参数指定 DataType、ParamType 及 Value.Type。由于每一个参数均出现多次，因此您只要设置其中一个参数即可。相关参数设置如表 18.20 所示。

表 18.20

参数名称	属性名称	属性设置值
Job_Id1	DataType	ftInteger
	ParamType	ptInput
	Value.Type	Byte
Job_Id2	DataType	ftInteger
	ParamType	ptInput
	Value.Type	Byte
Fname1	DataType	ftString
	ParamType	ptInput
	Value.Type	String
Fname2	DataType	ftString
	ParamType	ptInput
	Value.Type	String

Step 5 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str : String;
  Flag : Integer;
begin
   with Query1 do
   begin
     //先关闭
     Close;

     //更改 JobId1 的参数值
     if trim(EditJobid1.text) = '' then
        Params[0].value := null
```

```
      else
        Params[0].value := StrToInt(EditJobId1.text);
      //更改 JobId2 的参数值
      if trim(EditJobid2.text) = '' then
        Params[3].value := null
      else
        Params[3].value := StrToInt(EditJobId2.text);
      //更改 Name1 的参数值
      if trim(EditName1.text) = '' then
        Params[6].value := null
      else
        Params[6].value := EditName1.text;
      //更改 Name2 的参数值
      if trim(EditName2.text) = '' then
        Params[9].value := null
      else
        Params[9].value := EditName2.text;
      {通过 BDE 作最优化的处理}
      Prepare;
      //打开
      Open;
    end;
  end;
```

6 保存工程。

8 按【F9】功能键运行工程。工程运行后，您可以任意输入查询条件。图 18.18 以查询职务编号(Job Id)介于 5~15 之间，而且员工姓名(First Name)是介于字母 C~E 之间的员工数据为例。

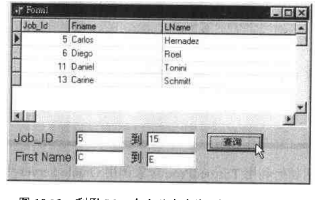


图 18.18　利用 SQL 命令动态查询(以 MS SQL 为例)

18.4 TStoredProc 对象

TStoredProc 对象除了 TTable 对象与 TQuery 对象外，另一个常用的是 TDataSet 对象。TStoredProc 对象允许您调用已经存在数据库中的 Stored Procedure(存储过程)，通过 Stored Procedure 访问数据除了可以减少网络上的传输，更能提高运行速度。一般来说，通过 TStoredProc 对象来调用数据库中的 Stored Procedure 之前，您除了要设置连接外(参考第 17 章)，还必须事先在数据库上创建 Stored Procedure。至于如何编写 Stored Procedure，请您参考其他的数据库书籍，本书仅示范几个常用的主题。以下各小节将示范如何通过 TStoredProc 对象来访问 Oracle 8i Database Server 与 Microsoft SQL Server 的各种类型的 Stored Procedure。

18.4.1 TStoredProc 对象访问 Oracle 数据库

18.4.1.1 Oracle 数据库的 Stored Procedure

Stored Procedure(存储过程)与 Delphi 6 的 Procedure(过程)在用法上是非常相近的。如果您想通过 Delphi 6 的 TStoredProc 对象来调用 Oracle Stored Procedure，除了要先设置连接外，还必须完成下列几个步骤。

步骤一：在 Oracle Database Server 端创建 Stored Procedure。

步骤二：在 Delphi 6 中利用 TStoredProc 对象进行连接测试。

1. 在 Oracle Database Server 端创建 Stored Procedure

要创建 Stored Procedure 您可以利用 Oracle Procedure Builder、Oracle Developer、Oracle Designer 等 Oracle 公司的前台开发软件。如果您手边没有这些软件，您也可以用 Oracle SQL*Plus 来创建 Stored Procedure。我们通常利用 Stored Procedure 来运行一些固定的工作，以下将示范如何进行员工年度调薪。这个 Stored Procedure 有一个参数，它代表调薪的幅度。其内容如下：

```
Create or Replace PROCEDURE RaiseSalary(RaiseRate IN NUMBER) IS   BEGIN
UPDATE emp   SET Sal = Sal * (1+RaiseRate);
END;
```

2. 在 Delphi 6 中利用 TStoredProc 对象进行连接测试

18-16

目的：本范例主要说明如何利用 TStroedProc 对象调用 Oracle Server 中的 Stroed Procedure。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TDatabase 对象、一个 TStoredProc 对象、一个 TLabel 对象、一个 TEdit 对象及两个 TButton 对象其相关属性设置如表 18.21 所示。

表 18.21

对象	属性名称	属性设置值
TDatabase	Name	Database1
	Alias	MyOracle
	DatabaseName	abc
	Params	UserName=scott Password=tiger
	LoginPrompt	False
	HandleShared	True
	connected	False
TStoredProc	Name	StoredProc1
	DatabaseName	abc
	StoredProcName	RAISESALARY
TStoredProc.Params[0]	DataType	ftFloat
	ParamType	ptInput
TLabel	Name	Label1
	Caption	调薪比率(输入参数):
TEdit	Name	Edit1
TButton	Name	Button1
	Caption	调用 Stored Procedure
	Enabled	False
TButton	Name	Button2
	Caption	数据库连接

步骤 3　在 Button2(数据库连接)对象的 onClick 事件处理过程中加入以下的代码:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //进行数据库连接
  Try
    Database1.Open;
  Finally
    ShowMessage('数据库连接成功!!');
    Button1.enabled := True;
    Button2.enabled := False;
  end;
end;
```

步骤 4　在 Button1(调用 Stored Procedure)对象的 onClick 事件处理过程中加入以下的代码:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    //打开 Stored Procedure
    Try
     StoredProc1.Params[0].AsFloat:=StrToFloat(Edit1.Text);
     StoredProc1.Prepare;
     StoredProc1.ExecProc;
    Finally
     ShowMessage('数据处理完毕!!');
    end;
end;
```

5 保存工程。

6 按【F9】功能键运行工程。工程运行后您可以先单击“数据库连接”按钮，并输入调职的比率，最后再单击“确定”按钮。运行后您可能无法看到结果，此时您可以利用 SQL*Plus 登录 Oracle Server，加以检查运行结果，如图 18.19 所示。

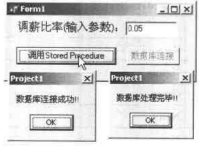


图 18.19 使用 TStoredProc 对象调用 Stroed Procedure

18.4.1.2 Oracle 数据库的 Stored Function

Stored Function(存储函数)与 Delphi 6 的 Function(函数)在用法上是非常相近的。一般来说，Stored Function 与 Stored Procedure 最大的不同在于，前者可以有返回值(Return)。事实上，您也可以设置 Stored Procedure 的参数为 OUT 或 INOUT，让这个 Stored Procedure 具有返回值的功能。如果您想通过 Delphi 6 的 TStoredProc 对象来调用 Oracle Stored Function，除了要先设置连接外，还必须完成下列几个步骤：

步骤一：在 Oracle Database Server 端创建 Stored Function。

步骤二：在 Delphi 6 中利用 TStoredProc 对象进行连接测试。

1. 在 Oracle Database Server 端创建 Stored Funcation

要创建 Stored Function，您可以利用 Oracle Procedure Builder、Oracle Developer、Oracle Designer 等 Oracle 公司的前台开发软件。如果您手边没有这些软件，您也可以用 Oracle SQL*Plus 来创建 Stored Function。我们通常利用 Stored Function 来运行一些计算或统计性的工作，以下将示范如何利用 Stored Function 来获取某个部门的平均工资。这个 Stored Function 有一个参数与一个返回值，参数代表部门代号，返回值代表这个部门的平均工资。其内容如下：

```
Create or Replace FUNCTION SumDeptSalary(Dept_No IN NUMBER)
RETURN number IS
      Total   NUMBER(11,2);
BEGIN
      SELECT   Sum(sal) INTO  Total
         FROM      emp
         WHERE   DEPTNO = Dept_No;
      RETURN(Total);
END;
```

2. 在 Delphi 6 中利用 TStoredProc 对象进行连接测试



18-17

目的：本范例主要说明如何利用 TStroedProc 对象调用 Oracle Server 中的 Stroed Function。

(1) 创建新的工程。

(2) 在 TForm 对象中加入一个 TDatabase 对象、一个 TStoredProc 对象、两个 TLabel 对象、两个 TEdit 对象及两个 TButton 对象其相关属性设置如表 18.22 所示。

表 18.22

对象	属性名称	属性设置值
TDatabase	Name	Database1
	Alias	MyOracle
	DatabaseName	abc
	Params	UserName=scott
		Password=tiger
	LoginPrompt	False
	HandleShared	True
	connected	False
TStoredProc	Name	StoredProc1
	DatabaseName	abc
	StoredProcName	SUMDEPTSALARY
TStoredProc.Params[0]	DataType	ftFloat
	ParamType	ptResult
TStoredProc.Params[1]	DataType	ftFloat
	ParamType	ptInput
TLabel	Name	Label1
	Caption	部门代号(输入参数):
TLabel	Name	Label2
	Caption	工资总额(返回值):

续上表

对象	属性名称	属性设置值
TEdit	Name	EditDeptID
TEdit	Name	EditSalary
TButton	Name	Button1
	Caption	调用 Stored Function
	Enabled	False
TButton	Name	Button2
	Caption	数据库连接

STEP 3 在 Button2(数据库连接)对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Try
    Database1.Open;
  Finally
    ShowMessage('数据库连接成功!');
    Button1.Enabled := True;
    Button2.Enabled := False;
  end;
end;
```

STEP 4 在 Button1(调用 Stored Function)对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Try
   StoredProc1.Params[1].AsFloat:=StrToInt(EditDeptID.Text);
   StoredProc1.Prepare;
   StoredProc1.ExecProc;
   EditSalary.Text:=FloatToStr(StoredProc1.Params[0].AsFloat);
  Finally
   ShowMessage('数据处理完毕!!');
  end;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程。工程运行后您可以先单击“数据库连接”按钮，并输入部门代号，最后，再单击“确定”按钮，此时系统会将指定部门的工资总额加以返回，如图 18.20 所示。



图 18.20　使用 TStoredProc 对象调用 Stroed Function

18.4.1.3　*Oracle 数据库的 Stored Package*

在 Oracle Database Server 上为了有效管理 Stored Function(存储函数)与 Stored Procedure(存储过程)我们通常会将这些 Stored Function(存储函数)与 Stored Procedure(存储过程)打包成 Package 的形式。Package 除了可以有效管理 Stored Function(存储函数)与 Stored Procedure(存储过程)外，另外有一个非常重要的用途，那就是返回查询的 Cursor，如果您对于 Cursor 这个名词不是很了解的话，那您可以将它想象成是一个符合查询结果的数据集(Result Set)。换言之，Package 可以为您返回一条或多条的数据记录(Record)，而不再只是运行某个操作，或返回某个值而已。

当笔者以 TStoredProc 对象进行连接 Oracle Database Server 的 Package 时，发现一个非常糟糕的现象，那就是 TStoredProc 对象无法读取 Package(如果您可以读取可别忘了要通知我一下喔!!)。这样的现象就表示原本已编写的 Package 就无法使用，其实这只要花点时间再将 Package 改写成 Stored Function(存储函数)与 Stored Procedure(存储过程)就可以了。但至于返回查询的 Cursor，可就无法解决了，因为您无法单独使用 Stored Function(存储函数)与 Stored Procedure(存储过程)来返回 Cursor。不过呢！山不转路转，我们还是有方法可以解决的。我们先来说明 Package 如何返回 Cursor。以下是一般的写法。

步骤一、先创建 Package Specification

```
CREATE OR REPLACE PACKAGE MyPackage IS
  TYPE MyCur IS REF CURSOR RETURN emp%ROWTYPE;
  PROCEDURE PackageSP(v_DeptNo IN NUMBER , curl lN OUT MyCur);
END;
```

步骤二、再创建 Package Body

```
CREATE OR REPLACE PACKAGE BODY  MyPackage IS
Procedure PackageSP(v_DeptNo IN NUMBER , curl IN OUT MyCur) IS
BEGIN
      OPEN curl FOR
            SELECT * FROM emp  WHERE DeptNo=v_DeptNo;
END;
END;
```

以下是我们的改良方法。

```
目的：调用 Stored Procedure 并返回 Cursor 的 Result Set
步骤一、先声明一个 Package specification，其中的 emp 代表所指定的 Table Name。
CREATE OR REPLACE PACKAGE aCurType IS
    TYPE MyCur IS REF CURSOR RETURN emp%ROWTYPE;
```

```
END;
步骤二、再创建 Stored Procedure，并设置 Cursor 的参数类型为 IN OUT，而且其 Type 设为
Package specification 中所设置的 Type。这个 Stored Procedure 的目的在返回指定部门的
所有数据，其中 emp 是数据表名称(Table Name)，而 DeptNo 是代表部门的字段名称。
CREATE OR REPLACE PROCEDURE PackageSP(v_DeptNo IN NUMBER , curl IN OUT
aCurType.MyCur) IS
BEGIN
      OPEN curl FOR
            SELECT * FROM emp  WHERE DeptNo=v_DeptNo;
END;
```

要利用 TStoredProc 对象来调用我们改良的 Stored Procedure，除了要先设置连接外(参考 17 章)，还必须完成下列几个步骤。

步骤一：在 Oracle Database Server 端创建改良的 Stored Procedure。

步骤二：在 Delphi 6 中利用 TStoredProc 对象进行连接测试。

1. 在 Oracle Database Server 端创建 Stored Procedure

要创建 Stored Procedure 您可以利用 Oracle Procedure Builder、Oracle Developer、Oracle Designer 等 Oracle 公司的前台开发软件。如果您手边没有这些软件，您也可以用 Oracle SQL*Plus 来创建我们所改良的 Stored Procedure。我们这个 Stored Procedure 主要是返回指定部门代号的所有员工数据，至于内容请参考本小节前面所述。

2. 在 Delphi 6 中利用 TStoredProc 对象进行连接测试



18-18

目的：本范例主要说明如何利用 TStroedProc 对象调用 Oracle Server 中的 Stroed Procedure，并返回查询的 Cursor Result Set。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TDatabase 对象、一个 TStoredProc 对象、一个 TDataSource 对象、一个 TDBGrid 对象、一个 TLabel 对象、一个 TEdit 对象及两个 TButton 对象其相关属性设置如表 18.23 所示。

表 18.23

对象	属性名称	属性设置值
TDatabase	Name	Database1
	Alias	MyOracle
	DatabaseName	abc
	Params	UserName=scott Password=tiger
	LoginPrompt	False
	HandleShared	True
	connected	False

续上表

对象	属性名称	属性设置值
TStoredProc	Name	StoredProc1
	DatabaseName	abc
	StoredProcName	PACKAGESP
TStoredProc.Params[0]	DataType	ftFloat
	ParamType	ptInput
	Value	10
TStoredProc.Params[1]	DataType	ftCursor
	ParamType	ptInputOutput
TDataSoucre	Name	DataSource1
	DataSet	StoredProc1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TLabel	Name	Label1
	Caption	部门代号(输入参数)：
TEdit	Name	EditDeptID
TButton	Name	Button1
	Caption	调用 Stored Package
	Enabled	False
TButton	Name	Button2
	Caption	数据库连接

(3) 如果您在步骤 2 正确的设置各项属性值，您只要再将 Database1 的 Connected 属性及 StoredProc1 的 Active 属性设置 True。那么您将会发现 TDBGrid 会显示出部门代号为 10 的所有员工数据，如图 18.21 所示。为什么是部门 10 呢?那是因为我们 TStoredProc.Params[0]的 Value 值设为 10 的缘故。但是这里请先将 Database1 的 Connected 属性及 StoredProc1 的 Active 属性设置 False，因为我们要示范动态数据库连接与数据访问。

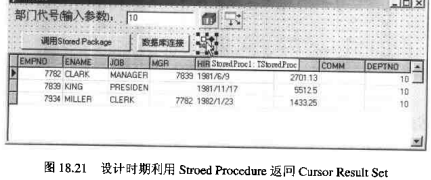


图 18.21　设计时期利用 Stroed Procedure 返回 Cursor Result Set

Step 4 在 Button2(数据库连接)对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Try
    Database1.Open;
  Finally
    ShowMessage('数据库连接成功!!');
    Button2.Enabled := False;
    Button1.Enabled := True;
  end;
end;
```

Step 5 在 Button1(调用 Stored Package)对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Try
    with StoredProc1 do
    begin
      Active := False;
      //设定参数值
      Params[0].value:=StrToInt(EditDeptID.Text);
      Active := True;
    end;
  Finally
   ShowMessage('数据处理完毕!!');
  end;
end;
```

Step 6 保存工程。

Step 7 按【F9】功能键运行工程。工程运行后您可以输入任何指定的部门代号，以返回该部门的所有员工数据，如图 18.22 所示。

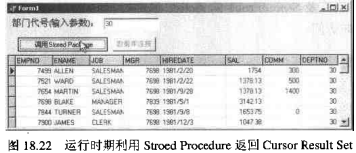


图 18.22 运行时期利用 Stroed Procedure 返回 Cursor Result Set

18.4.2 TStoredProc 对象访问 MS SQL 数据库

Microsoft SQL Server 所提供的 Stored Procedure(存储过程)比起 Oracle Database Server 要

简单多了，但其运行效率呢?这个嘛！请大家自己比较评估，笔者在此不做任何建议。以下将以二个小节对应说明如何利用 Microsoft SQL Server 制作出与 Oracle Database Server 所提供的 Stored Procedure、Stored Function 及 Stored Package 相同的功能。

18.4.2.1　MS SQL 数据库的 Stored Procedure(一)

Stored Procedure 特性可能有输入参数或没有输入参数，但一定没有返回值。因此如果需要返回值就必须将参数设置输出参数(OUT)，关于这个部分请参考下一小节。要利用 TStoredProc 对象来调用 MS SQL Server 的 Stored Procedure，除了要先设置连接外(参考第 17 章)，还必须完成下列几个步骤。

步骤一：在 Microsoft SQL Server 端创建 Stored Procedure。

步骤二：在 Delphi 6 中利用 TStoredProc 对象进行连接测试。

1. 在 Microsoft SQL Server 端创建 Stored Procedure

要创建 Stored Procedure 您可以利用 Query Analyer 或 Enterprise Manager 来完成，如何编写请参考 SQL Server 相关书籍，以下仅列出删除指定职务代号(Job_Id)的所有员工的相关 SQL 语句。

```
CREATE PROCEDURE DeleteEmployee @InJobID   int AS
   DELETE   Employee where Job_Id = @InJobID
```

2. 在 Delphi 6 中利用 TStoredProc 对象进行连接测试：



18-19

目的：本范例主要说明如何利用 TStroedProc 对象调用 Microsoft SQL Server 中的 Stroed Procedure。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TDatabase 对象、一个 TStoredProc 对象、一个 TLabel 对象、一个 TEdit 对象及一个 TButton 对象其相关属性设置如表 18.24 所示。

表 18.24

对象	属性名称	属性设置值
TDatabase	Name	Database1
	Alias	MyMSSQL
	DatabaseName	abc
	Params	UserName=sa Password=
	LoginPrompt	False
	HandleShared	True
	connected	True

续上表

对象	属性名称	属性设置值
TStoredProc	Name	StoredProc1
	DatabaseName	abc
	StoredProcName	dbo.DeleteEmployee
TStoredProc.Params[0]	DataType	ftInteger
	ParamType	ptInput
	Value.Type	Byte
TLabel	Name	Label1
	Caption	删除指定的 Job_ID
TEdit	Name	Edit1
TButton	Name	Button1
	Caption	调用 Stored Procedure

3 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //调用 Stored Procedure
  Try
   StoredProc1.Params[1].AsInteger:=StrToInt(Edit1.Text);
   StoredProc1.Prepare;
   StoredProc1.ExecProc;
  Finally
   ShowMessage('数据处理完毕!!');
  end;
end;
```

4 保存工程。

5 按【F9】功能键运行工程。工程运行后您可以输入任何指定的职务代号(Job ID)，再单击“调用 Stored Procedure”以删除所属的员工数据，如图 18.23 所示。

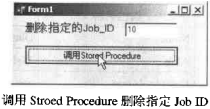


图 18.23 调用 Stroed Procedure 删除指定 Job ID 的员工数据

18.4.2.2 MS SQL 数据库的 Stored Procedure(二)

前一小节，我们示范如何调用一个没有返回值的 Stored Procedure，本小节将更进一步地说明如何将参数设置为输出参数(OUT)，以达到返回值的目的。要利用 TStoredProc 对象来调

用 MS SQL Server 的 Stored Procedure，除了要先设置连接外(参考第 13 章)，还必须完成下列几个步骤。

步骤一：在 Microsoft SQL Server 端创建 Stored Procedure。

步骤二：在 Delphi 6 中利用 TStoredProc 对象进行连接测试。

一、在 *Microsoft SQL Server* 端创建 *Stored Procedure*

要创建 Stored Procedure 您可以利用 Query Analyer 或 Enterprise Manager 来完成，如何编写请参考 SQL Server 相关书籍，以下仅列出如何获取指定职务代号(Job_Id)的所有员工人数。

```
CREATE PROCEDURE GetEmployeeCount @InJobId Int, @OutCount Int OUT   AS
  SELECT   @OutCount = Count(*)
  FROM employee
  WHERE Job_Id = @InJobID
```

2. 在 *Delphi 6* 中利用 *TStoredProc* 对象进行连接测试



18-20

目的：本范例主要说明如何利用 TStroedProc 对象调用 Microsoft SQL Server 中的 Stroed Procedure。本范例修改自范例 18-19，因此相同的设置不再重复说明，以下仅列出不同的对象设置。

步骤 1　在 TForm 对象上再加入一个 TLabel 对象(Label2)及一个 TEdit 对象(EditCount)，并将新的 Label2 的 Caption 属性设置为“此 Job_ID 的总人数”，而且将原来 Label1 的 Caption 属性设置为“查询的 Job_ID”。

步骤 2　将 StoredProc1 对象的 StoredProcedureName 设为“dbo.GetEmployee Count”，并将其 Params 属性值设置如表 18.25 所示。

表 18.25

对象	属性名称	属性设置值
TStoredProc.Params[0]	DataType	ftInteger
	ParamType	ptInput
	Value.Type	Byte
TStoredProc.Params[1]	DataType	ftInteger
	ParamType	ptOutput
	Value.Type	Byte

步骤 3　在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Try
   StoredProc1.Params[1].value:=StrToInt(EditJobID.Text);
```

```
    StoredProc1.Prepare;
    StoredProc1.ExecProc;
    EditCount.Text:= IntToStr(StoredProc1.Params[2].value);
   Finally
    ShowMessage('数据处理完毕!!');
   end;
  end;
```

步骤 4 保存工程。

步骤 5 按【F9】功能键运行工程。工程运行后，您可以输入已经访问的职务代号(Job Id)，那么系统就会响应该职务代表的所有员工人数，如图 18.24 所示。



图 18.24 获取 Stroed Procedure 的返回值

18.4.2.3 MS SQL 数据库的 Stored Procedure(三)

前两个小节中，我们示范如何调用有返回值与没有返回值的 Stored Procedure。本小节将为大家说明如何更进一步获取查询结果。要利用 TStoredProc 对象来调用 MS SQL Server 的 Stored Procedure，除了要先设置连接外(参考第 17 章)，还必须完成下列几个步骤。

步骤一：在 Microsoft SQL Server 端创建 Stored Procedure。

步骤二：在 Delphi 6 中利用 TStoredProc 对象进行连接测试。

1. 在 Microsoft SQL Server 端创建 Stored Procedure

要创建 Stored Procedure 您可以利用 Query Analyer 或 Enterprise Manager 来完成，如何编写请参考 SQL Server 相关书籍，以下仅列出如何获取指定职务代号(Job_Id)的所有员工数据。

```
CREATE PROCEDURE  GetEmployee2 @InJobID int  AS
  SELECT  Job_Id,Emp_Id,FName,Hire_date
  FROM  employee
  WHERE Job_Id = @InJobId
```

2. 在 Delphi 6 中利用 TStoredProc 对象进行连接测试

范例 18-21

目的：本范例主要说明如何利用 TStroedProc 对象获取 Microsoft SQL Server 中 Stroed Procedure 的查询结果。本范例修改自范例 18-19，因此相同的设置不再重复说明，以下仅列出不同的对象设置。

步骤 1 将 TForm 对象原来 Label1 的 Caption 属性设置为“查询的 Job_ID”。

(2) 将 StoredProc1 对象的 StoredProcedureName 设为“dbo.Get Employee2”，而且将其 Params 属性值设置如表 18.26 所示。

表 18.26

对象	属性名称	属性设置值
TStoredProc.Params[0]	DataType	ftInteger
	ParamType	pdInput
	Value.Type	Byte

(3) 在 TForm 对象中加入一个 TDBGrid 对象，并将其 DataSet 属性设为“StoredProc1”。

(4) 将原来 Button1 的 Caption 属性设为“调用 Stored Package”，并在其 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Try
    with StoredProc1 do
    begin
      //指定参数值
      Params[1].value := StrToInt(EditjobiD.Text);
      StoredProc1.Prepare;
      StoredProc1.Active := True;
    end;
  Finally
   ShowMessage('数据处理完毕!!');
  end;
end;
```

(5) 保存工程。

(6) 按【F9】功能键运行工程。工程运行后，您可以输入指定的职务代号(Job Id)并单击“调用 Stored Package”按钮，那么系统就会返回该职务代号的所有员工数据，如图 18.25 所示。

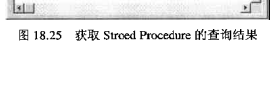


图 18.25　获取 Stroed Procedure 的查询结果

第19章

Delphi 6 数据库程序设计(三)

本章目标

在前面两章中，我们介绍了有关数据库的基本概念及一些常用的数据库对象。本章除了继续讨论 Delphi 6 其他常用的数据库对象外，另外为大家介绍 Delphi 6 数据库程序设计其他重要的课题，包括报表的制作及安装程序的应用。

本章主题

19.1 数据库程序设计常用的对象

19.2 Delphi 6 的报表对象

19.3 Delphi 6 的安装程序

19.1 数据库程序设计常用的对象

本小节将为大家介绍 Delphi 6 数据库程序设计中常用的对象，包括 TDataMoudle、TDatabase、TDBGrid 等，分述如下。

19.1.1 TDataModule 对象

前面的章节中曾经使用 TDataModule 对象。TDataModule 对象我们可以把它当作是一个隐形的 Form 对象，它与 Form 对象一样可以加入对象，但通常仅限于非可见组件(Non-Visual Component)。使用 TDataModule 对象最大的好处就是减少资源的浪费(整个工程的非可见组件可以共享)及便于管理。以下将通过实例来说明如何使用 TDataModule 对象。

范例 19-1

目的：本范例主要说明如何使用 TDataModule 对象。

(1) 创建新的工程。

(2) 您可以有两个方法为工程添加 TDataModule。第一个方法请选择下拉式菜单“File/New/Data Module”；第二个方法是从“File/New/Other...”下拉式菜单打开“New Items”对话框，并且在“New”页标签中选取“DataModule”，最后再单击“OK”按钮，以为工程加入 TDataModule 对象，如图 19.1 所示。

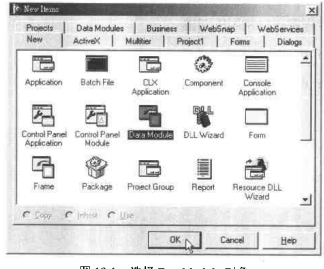


图 19.1 选择 DataModule 对象

(3) 在 DataModule2(上一步骤所产生的 TDataModule)中加入一个 TTable 对象与 TDataSource 对象，其相关属性设置如表 19.1 所示。

表 19.1

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	EMPLOYEE.DB
	IndexFieldName	EmpNo
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1

4 因为 TTable 与 TDataSource 对象是放在 DataModule 对象中，因此 TForm 对象并不认识它们，因此请点选“File/Use Unit…”下拉式菜单。此时出现“Use Unit”对话框，再点选“Unit2”，并单击“OK”按钮。如图 19.2 所示。

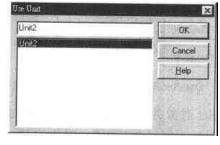


图 19.2　选择 Use Unit

5 通过上一步骤的设置，TForm 对象中的数据控制组件(如 TDBGrid 对象)已经可以直接访问 DataModule2 的 TTable 或 TDataSource 对象了。所在请在 TForm 对象中加入一个 TDBGrid 对象及 TDBNavigator 对象，其相关属性设置如表 19.2 所示。

表 19.2

对象	属性名称	属性设置值
TDBGrid	Name	DBGrid1
	DataSource	DataModule2.DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataModule2.DataSource1

6 保存工程。

7 按【F9】功能键运行工程。工程运行后，您会发现有没有使用 TDataModule 对象对用户而言是没有什么不同的。但对设计者而言，却有相当多的好处，如图 19.3 所示。

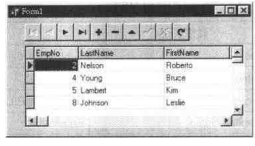


图 19.3　使用 DataModule 的运行结果

19.1.2　*TDatabase* 对象

在前面章节示范与 MS SQL Server 或 Oracle Server 的连接中曾经使用 TDatabase 对象。我们这里来谈谈 TDataSet 对象与 TDatabase 对象之间的关系，如图 19.4 所示。

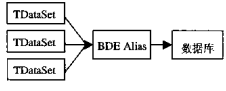


图 19.4　DataSet 未通过 TDatabase 对象读取数据库

在图 19.4 中是显示 TDataSet 对象并没有通过 TDatabase 对象来访问数据库。因此，一旦 BDE Alias 有更改时，必须同时修改每一个 DataSet 对象的 DatabaseName 属性。再者，如果其所连接的数据库需要登录(Login)时，您可能需要分别对所有的 DataSet 对象进行设置。

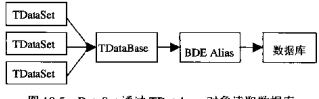


图 19.5　DataSet 通过 TDatabase 对象读取数据库

在图 19.5 中是显示 TDataSet 对象通过 TDatabase 对象来访问数据库。因此就算 BDE Alias 有更改，我们只需修改 TDatabase 对象中的 AliasName 属性即可。再者，如果其所连接的数据库需要登录(Login)时，我们只要对 TDatabase 对象进行设置就可以。TDatabase 对象除了有上述的好处外，其最大的好处就是能与数据库保持连接，以减少访问时间，提高运行效率。

在前几章的范例中，我们也曾经说明 TDataBase 对象常用的属性，因此不再赘述，这里以表 19.3 做简要说明。

表 19.3

TDatabase 属性名称	说明
AliasName	代表所使用的 BDE 别名(Alias)
Connected	代表与数据的连接状态，当 Connected 属性值为 True，表示已经与数据库获取连接

续上表

TDatabase 属性名称	说明
KeepConnection	代表是否与数据库保持连接，当 KeepConnection 属性值为 True，表示就算没有任何一个 DataSet 对象与数据库连接，TDatabase 对象也会保持与数据库连接的状态
DatabaeName	这个属性是用来代替实际的 Alias Name。如此一来，即使 BDE Alias 有更改，其他的 DataSet 对象完全不需做任何的修改
HandleShared	当 HandleShared 属性值为 True，可以避免远程的数据模块因相同名称而产生错误。
LoginPrompt	当 LoginPrompt 为 True，表示会出现数据库登录的对话框。反之，则不会出现。如果您要将 LoginPrompt 设为 Flase，请先在 Params 属性中输入 UserName 与 Password 的信息，否则无法正确地登录数据库
Params	此属性通常用来保存数据库登录的相关信息，如 UserName 与 Password 等。例如(以 Oracle 为例)： UserName=scott Password=tiger
DataSetCount	用来记录目前与 TDatabase 对象连接的 TDataSet 对象个数
DataSets	用来获取目前与 TDatabase 对象连接的 TDataSet 对象名称。例如 {找出目前与 TDatabase 连接的 TQuery 个数} C:=0; with Database1 do for I := 0 to DataSetCount - 1 do if DataSets[I] is TQuery then C:=C+1; ShowMessage(IntToStr(C));
ReadOnly	当 ReadOnly 属性值为 True，表示数据库的数据只允许读取，而无法插入与修改，反之，就可以读取、插入或修改

接着，为大家说明 TDatabase 常用的操作方法，分述如表 19.4 所示。

表 19.4

TDatabase 操作方法	说明
StartTransaction	打开一个事务(Transaction)，这里所谓的事务是数据库一个相当重要的名词。事务的概念允许您写回数据库的数据，以有恢复的机会。在您使用 Commit 命令确认之前，您都可以使用 Rollback 命令加以恢复
ApplyUpdates	将数据写回数据库
Commit	Commit 是用来确认事务(Transaction)。使用 Commit 命令之后，无法再使用 Rollback 命令加以恢复
Rollback	Rollback 是用来取消事务(Transcation)。Rollback 命令会将数据库的内容恢复到上一个 Commit 命令之后

范例 19-2

目的：本范例主要说明如何使用 TDatabase 对象中与事务(Transaction)有关的操作方法。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入一个 TDatabase 对象、一个 TTable 对象、一个 TDataSource 对象、一个 TDBGrid 对象、一个 TListBox 对象及四个 TButton 对象。其相关属性设置如表 19.5 所示。

表 19.5

对象	属性名称	属性设置值
TDatabase	AliasName	MyOracle
	Name	Database1
	DatabaseName	TestDB
	HandleShared	True
	LoginPrompt	False
	Params	UserName=Scott Password=tiger
	Connected	True
TTable	Name	Table1
	DatabaseName	TestDB
	TableName	EMP
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TListBox	Name	ListBox1
TButton	Name	btStartTransaction
	Caption	StartTransaction
TButton	Name	btDelete
	Caption	Delete
TButton	Name	btCommit
	Caption	Commit
TButton	Name	btRollback
	Caption	Rollback
TButton	Name	btTableRefresh
	Caption	TableRefresh

STEP 3 在 btStartTransaction 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btStartTransactionClick(Sender: TObject);
begin
  //打开一个事务
  Database1.StartTransaction;
  //显示事务开始
  ListBox1.Items.add('StartTransaction');
end;
```

STEP 4 在 btDelete 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btDeleteClick(Sender: TObject);
begin
  //记录删除
  ListBox1.Items.add(Table1.FieldByname('EMPNO').AsString
                     + ' 员工数据已被删除');
  //删除一条数据
  Table1.delete;
end;
```

STEP 5 在 btCommit 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btCommitClick(Sender: TObject);
begin
  //确认所有事务
  Database1.Commit;
  //显示恢复信息
  ListBox1.Items.add('Commit');
end;
```

STEP 6 在 btRollback 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btRollbackClick(Sender: TObject);
begin
  //恢复所有被删除的数据
  Database1.Rollback;
  //显示恢复信息
  ListBox1.Items.add('Roolback');
end;
```

STEP 7 在 btTableRefresh 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btTableRefreshClick(Sender: TObject);
begin
  //重新显示数据
  Table1.Refresh ;
  //回到第一条
  Table1.First;
end;
```

Step 8 保存工程。

Step 9 按下【F9】功能键运行工程。工程运行后，您可以依次运行 StartTransaction 按钮以打开一个事务、删除几条数据，并决定是否确认事务(Commit)，或是恢复(Rollback)所有修改的数据，最后再单击“Refresh”按钮以重新显示数据，如图 19.6 所示。

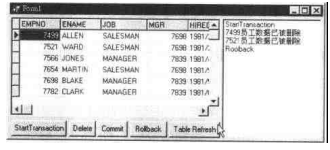


图 19.6 使用数据库对象的事务操作方法

19.1.3 TDBGrid 对象

TDBGrid 对象在 Delphi 6 数据库程序设计中也是经常被使用的对象。前面也有很多范例都曾经说明过。以下仅就 TDBGrid 对象常用的属性与事件处理加以说明。

19.1.3.1 TDBGrid 对象的常用属性

TDBGrid 对象有很多属性的用法与其他对象相同，以下仅就 TDBGrid 对象的“外观属性设置”、“计算性字段”及“Lookup 字段”等主题加以说明。

外观属性设置

Columns 属性继承自 TDBGridColumns 对象，其代表 TDBGrid 对象所显示的每个数据字段。您只要在 Colums 属性字段双击，或是在 TDBGrid 对象上按右键选择“Colums Editor...”都可打开字段编辑器(Columns Editor)。通过字段编辑器，您可以直接来设置 TDBGrid 对象的每个显示字段的外观属性。以下几个步骤将示范如何设置 TDBGrid 对象的每个显示字段的外观属性。

Step 1 假设您已经在 TForm 对象中加入 TTable 对象、TDataSource 及 TDBGrid 对象，而且设置好这些对象间的关系。

Step 2 接着在“TDBGrid”对象上按右键，并选择“Columns Editor”，如图 19.7 所示。

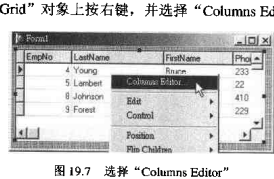


图 19.7 选择“Columns Editor”

③ 此时“Columns Editor”对话框会被打开。接着按右键，选择“Add All Fields”。请特别注意此时 TDBGrid 所对应的 TDataSet 必须是打开的，您才能获取所有的字段，如图 19.8 所示。



图 19.8　选择“Add All Fields”

④ 此时系统会将 TTable 对象上所有的字段名称全部加以载入。您只要点选某个 Column，就可以在“对象检查器”中直接设置相关属性。您也可以在此删除或添加数据字段，如图 19.9 所示。

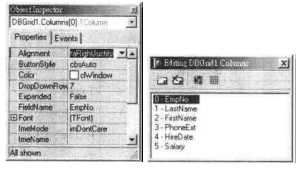


图 19.9　设置每个字段(Column)的属性

表 19.6 说明与 TDBGrid 对象 Column 有关的属性。

表 19.6

Column 的属性	说明
Title	与每个字段标题有关的属性
Title.Alignment	字段标题文字的对齐方式，包括 taLeftJustify、 taRightJustify 及 taCenter 等
Title.Caption	字段标题文字的内容
Title.Color	字段标题的背景颜色
Title.Font	与字段标题文字有关的属性设置
Alignment	此字段内容的对齐方式，包括 taLeftJustify、 taRightJustify 及 taCenter 等
FieldName	与 TDataSet 连接的字段名称
ReadOnly	此字段内容是否为只读，当 ReadOnly 属性值为 True，表示字段内容是只读。反之，可修改
Width	每个字段显示的宽度

计算性字段

所谓“计算性字段”是指在数据表格中并没有存在这个字段，它可能是由其他字段经计算后而产生的。事实上，计算性字段是无法单独依靠 TDBGrid 对象来完成的，我们通常在 TTable 进行相关设置，再由 TDBGrid 对象负责显示。

LookUp 字段

所谓“LookUp”是指在数据表中并没有存在这个字段，它可能是由其他 Dataset 而产生的。事实上，Lookup 字段是无法单独依靠 TDBGrid 对象来完成的，我们通常在 TTable 进行相关设置，再由 TDBGrid 对象负责显示。

范例 19-3

目的：本范例主要说明如何利用 TTable 与 TDBGrid 对象来完成计算性字段与 Lookup 字段。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入两个 TTable 对象、一个 TDataSource 对象及一个 TDBGrid 对象。其相关属性设置如表 19.7 所示。

表 19.7

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	Parts.db
	Active	True
TTable	Name	Table2
	DatabaseName	DBDEMOS
	TableName	vendors.db
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1

注意

本范例使用 DBDEMOS 数据库别名中的 Parts.db 与 vendors.db 两个数据表(Table)。在 Parts.db 数据表中，我们有兴趣的字段包括 PartNo(产生编号)、VendorNo(供货商代号)、onHand(已存在数量)、onOrder(已订购数量)。而 vendors.db 数据表中，我们有兴趣的字段是 VendorNo(供货商编号)与 VendorName(供货商名称)。本范例将示范如何为 Table1(也就是对应于 Parts.db 的 TTable 对象)产生一个计算性字段与一个 Lookup 字段。计算性字段将显示 onHand 与 onOrder 两个字段的加总结果。而 Lookup 字段是利用 VendorNo 字段查找出 VendorName 字段值。

3 首先，我们先示范如何产生一个计算性字段。请在 Table1 对象上按右键，选择“Fields Editor”以打开“Fields Editor”，如图 19.10 所示。

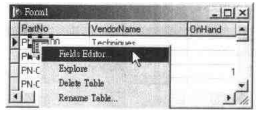


图 19.10　选择“Fields Editor”

4 接着，在“Fields Editor”对话框中，按右键选择“Add All Fields”，加入所有字段。并删除不必要的字段，最后仅留下 PartNo、VendorNo、onHand 与 onOrder 等字段，如图 19.11 所示。

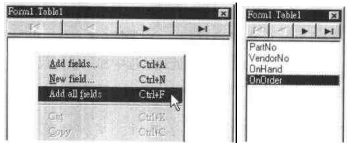


图 19.11　保留必要字段

5 此时，您可以通过“对象检查器”修改相关的属性。例如，您可以通过 DisplayFormat 属性来设置数据显示的格式(如 NT #,##0)；通过 CustomConstraint 属性来决定限制条件(如 Value > 0 and Value < 100)；通过 ConstraintErrorMessage 属性来决定当数据不符合限制条件的错误信息，如图 19.12 所示。

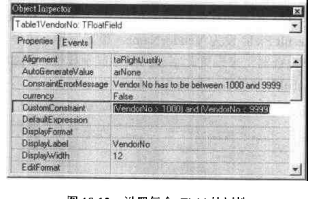


图 19.12　设置每个 Field 的属性

6 接着，我们要通过“Fields Editor”来新建一个“Total”的计算性字段，其字段值主要是由 onHand 字段与 onOrder 字段合计而得。请在“Fields Editor”对话框中按右键，选择“New Field..”。并将此字段名称设为 Total、type 设为 Integer、Field Type

设为 Calculated。设置完毕，请单击“OK”按钮。您可以更进一步地通过“对象检查器”来设置此字段的相关显示属性，例如“DisplayFormat”属性或“DisplayLabel”属性等，如图 19.13 所示。

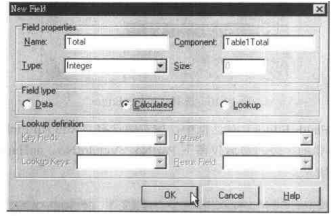


图 19.13 添加一个计算性字段

(7) 在 Table1 对象的 onCalcFields 事件处理过程中加入以下代码：

```
procedure TForm1.Table1CalcFields(DataSet: TDataSet);
begin
   DataSet.FieldByName('Total').Value :=
         DataSet.FieldByName('OnHand').Value +
         DataSet.FieldByName('OnOrder').Value ;
end;
```

(8) 接着，我们再通过“Fields Editor”来新建一个“VendorName”的 LookUp 字段，其字段值主要是通过 Table1 的 VendorNo 字段值来获取其所对应于 Table2 的 VendorName 字段值。请在“Fields Editor”对话框中按右键，选择“New Field..”。并将此字段名称设为 VerdorName、type 设为 String、Size 设为 20、DataType 设为 LookUp、Key Fields 设为 VendorNo、Dataset 设为 Table2、Lookup Keys 设为 VendorNo、Result Field 设为 VendorName。设置完毕，请单击“OK”按钮。您可以再通过“对象检查器”来设置此字段的其他显示属性，例如“DisplayFormat”属性或“DisplayLabel”属性等，如图 19.14 所示。

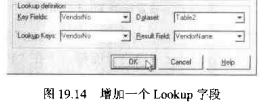


图 19.14 增加一个 Lookup 字段

9) 最后，请在 TDBGrid 对象上按右键，并选择“Columns Editor”。此时“Columns Editor”对话框会被打开。接着按右键，选择“Add All Fields”。此时系统会将 Table1 对象上所有的字段名称全部加以载入。这里请您删除 VendorNo 字段，并且利用鼠标拖动将 VendorName 字段移到 OnHand 字段之前，如图 19.15 所示。

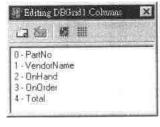


图 19.15　决定所有显示的数据字段

10) 保存工程。

11) 按【F9】功能键运行工程。工程运行后，“Total”字段与“VendorName”字段会依照公式的计算结果与其所对应的数据加以显示，如图 19.16 所示。

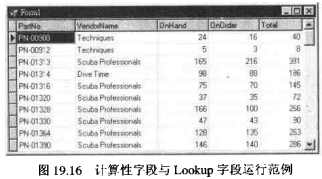


图 19.16　计算性字段与 Lookup 字段运行范例

19.1.3.2　TDBGrid 对象的常用事件

本小节将介绍 TDBGrid 对象的 onCellClick 事件、onColEnter 事件、onColExit 事件、onDblClick 事件、onEnter 事件、onExit 事件、onTitleClick 事件及 onDrawColumnCell 事件等。

- *onCellClick 事件*

当用户在 TDBGrid 对象的某一字段上，按一下鼠标左键即触发此事件，此事件的原型声明如下：

```
procedure TForm1.DBGrid1CellClick(Column: TColumn);
```

此事件处理过程有一个 Column 参数，其继承自 TColumn 对象。TColumn 对象代表 TDBGrid 对象的某个字段，关于 TColumn 对象的使用方法请参考其他相关章节或帮助文件，使用范例请参考范例 19-4。

- *onColEnter 事件*

当用户将输入焦点移至 TDBGrid 对象的某个单元格时则会触发 onColEnter 事件。

- *onColExit* 事件

当用户将输入焦点离开 TDBGrid 对象的某个单元格时则会触发 onColExit 事件。

- *onDblClick* 事件

当用户在 TDBGrid 对象上双击即触发此事件，此事件的原型声明如下：

```
procedure TForm1.DBGrid1DblClick(Sender: TObject);
```

- *onEnter* 事件

当用户将输入焦点移至 TDBGrid 对象时则会触发 onEnter 事件。

- *onExit* 事件

当用户将输入焦点离开 TDBGrid 对象时则会触发 onEnter 事件。

- *onTitleClick* 事件

当用户在 TDBGrid 对象的表头上按一下鼠标左键即触发此事件，此事件的原型声明如下：

```
procedure TForm1.DBGrid1TitleClick(Column: TColumn);
```

此事件处理过程有一个 Column 参数，其继承自 TColumn 对象。TColumn 对象代表 TDBGrid 对象的某个字段，关于 TColumn 对象的使用方法请参考其他相关章节或联机帮助文件，使用范例请参考范例 19-4。

- *onDrawColumnCell* 事件

当应用程序需要绘制 TDBGrid 对象的某个字段时会触发 onDrawColumnCell 事件，通常用于绘图或改变显示颜色时。其原型声明如下：

```
TDrawColumnCellEvent = procedure (Sender: TObject; const Rect:
        TRect; DataCol: Integer; Column: TColumn;
        State: TGridDrawState) of object;
property OnDrawColumnCell: TDrawColumnCellEvent;
```

onDrawColumnCell 事件继承自 TDrawColumnCellEvent，其包括 Sender 参数、Rect 参数、DataCol 参数、Column 参数及 State 参数。分述如下：

- Sender 参数代表触发此事件的对象，通常是对象本身。
- Rect 参数是一个只读的参数，其继承自 TRect 对象，代表显示的区域范围。
- DataCol 参数代表 Column 参数在所有 Columns 数组中索引顺序。
- Column 参数继承自 TColumn 对象，代表 TDBGrid 对象中的一个数据字段。
- State 参数继承自 TGridDrawState 对象，代表目前数据的状态，其包含 gdSelected、gdFocused 及 gdFixed 等可能值。gdSelected 代表目前的字段(cell)是被选取的、

gdFocused 代表目前的字段已经获取输入焦点(input focus)及 gdFixed 代表目前的字段是一个固定的范围。



19-4

目的：本范例主要说明如何使用 TDBGrid 对象的 onCellClick 事件及 onTitleClick 事件来获取目前的字段名称与字段值，及通过 onDrawColumnCell 事件改变指定数据范围的显示格式。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TListBox 对象、一个 TTable 对象、一个 TDataSource 及一个 TDBGrid 对象，其相关属性设置如表 19.8 所示。

表 19.8

对象	属性名称	属性设置值
Ttable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	animals.dbf
	Active	True
TdataSource	Name	DataSource1
	DataSet	Table1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TlistBox	Name	ListBox

步骤 3 在 TDBGrid 对象的 onCellClick 事件处理过程中加入以下的代码：

```
procedure TForm1.DBGrid1CellClick(Column: TColumn);
var
  MyVariant : Variant;
begin
  //获取目前字段的字段值
  MyVariant :=  Column.Field.Value;
  ListBox1.Items.Add('Click 的字段值:' + VarToStr(MyVariant));
end;
```

步骤 4 在 TDBGrid 对象的 onTitleClick 事件处理过程中加入以下的代码：

```
procedure TForm1.DBGrid1TitleClick(Column: TColumn);
var
  MyVariant : Variant;
begin
  //获取目前的字段名称
  ListBox1.Items.Add('目前字段名称:' + Column.Title.Caption);
```

```
  //获取目前字段的字段值
  MyVariant := Column.Field.Value;
  ListBox1.Items.Add('目前字段值:' + VarToStr(MyVariant));
end;
```

5 在 TDBGrid 对象的 onDrawColumnCell 事件处理过程中加入以下的代码：

```
procedure TForm1.DBGrid1DrawColumnCell(Sender: TObject;
      const Rect: TRect;DataCol: Integer; Column: TColumn;
      State: TGridDrawState);
begin
  if Table1.FieldByName('Size').AsFloat >= 30 then
    DBGrid1.Canvas.Font.Color := clBlue;
  DBGrid1.DefaultDrawDataCell(Rect, Column.Field, State);
end;
```

6 保存工程。

7 按【F9】功能键运行工程。工程运行后，您会发现当 Size 字段的值大于或等于 30 时，此条记录将会以蓝色来显示。而且当您 Click 某个单元格(Cell)或标题(Title)时，系统便会触发其所对应的事件，如图 19.17 所示。

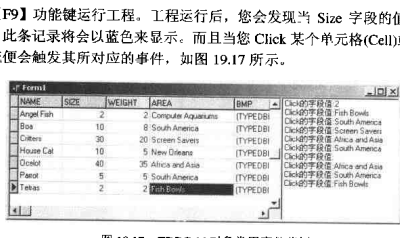


图 19.17 TDBGrid 对象常用事件范例

注意 onDrawColumnCell 事件除了可以利用 TDBGrid 对象的 Canvas 属性来改变字体显示颜色，Canvas 属性是继承自 TCanvas 对象，有关 TCanvas 对象请参考第 13 章的详细说明。

19.1.4 TDBCtrlGrid 对象

TDBCtrlGrid 对象与 TDBGrid 对象在功能上相当类似，它以表行的方式来显示，每一行中可以显示多个数据字段，而且一个 TDBCtrlGrid 对象中可以显示多个数据行。TDBCtrlGrid 对象必须配合 TDataSource 对象与其他数据感知控件一起使用，例如：TDBEdit 对象、TDBText 对象、TDBMemo 对象、TDBImage 对象、TDBListBox 对象、TDBComboBox 对象、TDBCheckBox 对象、TDBRadioGroup 对象、TDBLookupListBox 对象、TDBLookupComboBox 对象及 TDBRichEdit 对象等。由于这些数据感知控件与其他非数据感知控件在使用上相当接近，因此这里不在赘述。以下将针对 TDBCtrlGrid 对象常用的各种属性加以说明。

- *DataSource 属性*

DataSource 属性必须指定一个已经存在的 TDataSource 对象名称，与 TDBGrid 对象的 DataSource 属性相同。

- *AllowDelete 属性*

当 AllowDelete 属性值为 True，表示允许删除记录；反之，当其属性值为 False 时，则表示不允许删除记录，其默认值为 True。

- *AllowInsert 属性*

当 AllowInsert 属性值为 True，表示允许插入记录；反之，当其属性值为 False 时，则表示不允许插入记录，其默认值为 True。

- *ColCount 属性*

ColCount 属性值代表 TDBCtrlGrid 对象显示的列数，其默认值为 1。

- *Orientation 属性*

Orientation 属性值代表当 TDBCtrlGrid 对象显示的列数大于 1 时的显示顺序，Orientation 属性值为 goVertical，表示由上而下显示记录；Orientation 属性值为 goHorizontal，表示由左而右显示记录，其默认值为 goVertical。

- *PanelHeight 属性*

PanelHeight 属性值代表 TDBCtrlGrid 对象中每一个显示行的高度，通常会受 RowCount 属性值的影响而自动改变。

- *PanelWidth 属性*

PanelHeight 属性值代表 TDBCtrlGrid 对象中每一个显示行的宽度，通常会受 ColCount 属性值的影响而自动改变

- *RowCount 属性*

RowCount 属性值代表 TDBCtrlGrid 对象中的显示行个数。

- *ShowFocus 属性*

ShowFocus 属性值为 True 时，当输入焦点进入 TDBCtrlGrid 对象的显示行时，此显示列会出现一个虚框；反之，当 ShowFocus 属性值为 False 时，则显示行不会出现虚框。

- *SelectedColor 属性*

SelectedColor 属性值代表 TDBCtrlGrid 对象目前获取焦点的显示行的背景颜色。

- *Color 属性*

Color 属性值代表 TDBCtrlGrid 对象所有显示行的背景颜色。

范例 19-5

目的：本范例将示范如何使用 TDBCtrlGrid 对象来显示数据表中的内容。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象中加入一个 TTable 对象、一个 TDataSource 及一个 TDBCtrlGrid 对象，其相关属性设置如表 19.11 所示。

表 19.11

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	employee.db
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TDBCtrlGrid	Name	DBCtrlGrid1
	DataSource	DataSource1
	ColCount	3
	Color	clTeal
	RowCount	4
	SelectedColor	clYellow
	Align	alClient

STEP 3 在 TDBCtrlGrid 对象中加入两个 TLabel 对象及两个 TDBEdit 对象，其相关属性设置如表 19.10 所示。

表 19.10

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	FirstName
TLabel	Name	Label2
	Caption	HireDate
TDBEdit	Name	DBEdit1
	DataField	FirstName
TDBEdit	Name	DBEdit2
	DataField	HireDate

STEP 4 保存工程。

STEP 5 按【F9】功能键运行工程。工程运行后，您会发现总共出现了 12 个显示面板，只

有一个获取焦点的面版为黄色的背景色，如图 19.18 所示。

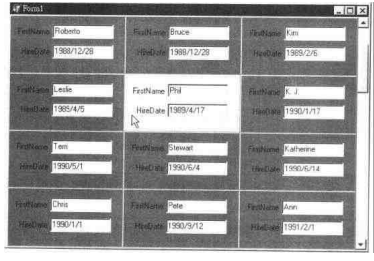


图 19.18　TDBCtrlGrid 对象运行范例

19.1.5　TDBNavigator 对象

TDBNavigator 对象提供一组控件，这些控件包括第一条记录(First)按钮、上一条(Prior)按钮、下一条(Next)按钮、最后一条(Last)按钮、插入(Insert)按钮、删除(Delete)按钮、编辑(Edit)按钮、写入数据库(Post)按钮、取消编辑(Cancel)按钮及刷新记录(Refresh)按钮，如图 19.19 所示。



图 19.19　TDBNavigator 对象示意图

表 19.11 说明 TDBNavigator 对象上每个按钮所代表的意义与目的。

表 19.11

按钮名称	意义与目的
第一条记录(First)按钮	调用 DataSet 的 First 操作方法，将目前记录指针设为第一条
上一条(Prior)按钮	调用 DataSet 的 Prior 操作方法，将目前记录指针设为上一条
下一条(Next)按钮	调用 DataSet 的 Next 操作方法，将目前记录指针设为下一条
最后一条(Last)按钮	调用 DataSet 的 Last 操作方法，将目前记录指针设为最后一条
插入(Insert)按钮	调用 DataSet 的 Insert 操作方法，在目前记录指针之前插入一个新的记录，并且将 DataSet 设为插入状态

续上表

按钮名称	意义与目的
删除(Delete)按钮	调用 DataSet 的 Delete 操作方法，删除目前的记录。假如 TDBNavigator 对象的 ConfirmDelete 属性值为 True，则在删除记录前会出现删除确认的对话框。反之，则直接删除
编辑(Edit)按钮	调用 DataSet 的 Edit 操作方法，将目前记录设为编辑(Edit)状态
写入数据库(Post)按钮	将目前记录中所做的修改写回数据库中，但其在多层数据库应用程序中，并非真正将数据写回数据库中，而且写回暂存内存中，待调用 ApplyUpdate 操作方法，才将数据真正的写回数据库
取消编辑(Cancel)按钮	调用 DataSet 的 Cancel 操作方法，放弃目前记录的编辑，并且将 DataSet 设为浏览状态
刷新记录(Refresh)按钮	清除 DataSet 中暂存显示的数据，再从数据库中重新获取数据

如果您不想使用 TDBNavigator 对象所有的按钮对象，您在设计时期可以通过对象检查器来设置它，如图 19.20 所示。

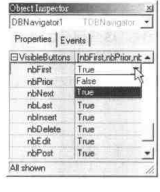


图 19.20 设计时期决定 TDBNavigator 对象显示的按钮

如果您必须在运行时期才能决定 TDBNavigator 对象所要显示的按钮，例如根据用户权限来决定显示的个数，那么您可以参考以下的代码。以下代码将为 TDBNavigator 对象显示第一条记录(First)按钮、上一条(Prior)按钮、下一条(Next)按钮及最后一条(Last)按钮等四个按钮。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  DBNavigator1.VisibleButtons := [nbFirst,nbPrior,nbNext,nbLast];
end;
```

19.2 Delphi 6 的报表对象

Delphi 6 提供了 QuickRep、QRSubDetial、QRStringBand、QRBand、QRChildBand、QRGroup、QRLabel、QRDBText、QRExpr、QRSysData、QRMemo、QRExprMemo、QRRichText、QRDBRichText、QRShape、QRImage、QRDBImage、QRCompositeReport、QRPreview、

QRTextFilter、QRCSVFilter、QRHTMLFilter 及 QRChart 等 23 个报表对象。基于篇幅的考虑，我们并不打算针对所有组件一一介绍。

19.2.1　Delphi 6 报表的概念

我们将以范例的方式来说明如何使用 Delphi 6 所提供的报表对象。在正式进入范例之前，我们先来谈谈 Delphi 6 报表的概念，如图 19.21 所示。

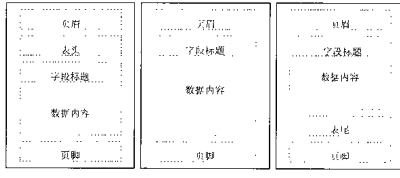


图 19.21　Delphi 6 的报表概念

我们利用上图来说明 Delphi 6 的报表概念，在图中有二张报表它们代表同一份报表。每一页都有“页眉”、“字段标题”、“数据内容”与“页脚”等不同的 Band，“表头”只会出现在第一页，而“表尾”只会出现在最后一页。有了以上的概念，我们来谈谈其所对应的报表对象，如表 19.12 所示。

表 19.12

报表概念名称	对应的报表对象(Band)	设置方法
报表主体	QuickRep	在 TForm 对象加入一个 TQuickRep 对象
页眉	PageHeaderBand	将 QuickRep.Bands.Has PageHeaderBand 属性设为 True
表头	TitleBand	将 QuickRep.Bands.Has TitleBand 属性设为 True
字段标题	ColumnHeaderBand	将 QuickRep.Bands.Has ColumnHeaderBand 属性设为 True
数据内容	DetailBand	将 QuickRep.Bands.Has DetailBand 属性设为 True。这个 Band 也是真正打印数据的地方，您只要在这个地方设置一次，则系统会重复打印，直到数据打印完毕为止
表尾	SummaryBand	将QuickRep.Bands.Has SummaryBand 属性设为 True
页脚	PageFooterBand	将 QuickRep.Bands.Has PageFooterBand 属性设为 True

上述的六种 Band 都必须配合其他报表对象才能完成。您可以根据不同的数据类型，而选择适当的报表对象来使用，每一个 Band 对象都有 BeforePrint 与 AfterPrint 等事件，分别在打印前、后被触发。

19.2.2 TQuickRep 对象的常用操作方法

以下就 TQuickRep(报表主体)常用的操作方法加以说明。

- *ExportFilter* 操作方法

ExportFilter 操作方法主要是将报表输出到文本文件(Text File)中。其使用方法如下所示：

```
QuickRep.ExportToFilter(TQRASCIIExportFilter.Create('文件名称.
                        Txt'));
[范例]
QuickRep.ExportToFilter(TQRASCIIExportFilter.Create('C:\a.txt');
```

- *NewPage* 操作方法

NewPage 操作方法可以强迫跳页打印。假如您想将“表尾”的结果打印到一个全新的页，那么您只要在 SummaryBand 对象的 BeforePrint 事件处理过程中加入以下的代码即可。

```
procedure TForm1.SummaryBand1BeforePrint(Sender:
                                    TQRCustomBand; var PrintBand: Boolean);
begin
  QuickRep1.NewPage;
end;
```

- *NewColumn* 操作方法

NewColumn 与 NewPage 操作方法最大的不同在于后者的对象是“页”，而前者的对象是“字段”。同样可以使用在 BeforePrint 与 AfterPrint 等事件处理过程中。

- *Preview* 操作方法

将 QuickRep 报表对象以预览的方式呈现。

- *Print* 操作方法

将 QuickRep 报表输出到打印机。

- *PrintSetup* 操作方法

打开打印机设置对话框，以提供给用户进行打印设置。

19.2.3 直列式(List)报表制作

直列式(List)报表是一种最简单的报表格式，也是经常被采用的一个格式。本小节将为大家示范如何制作直列式报表，在报表中，我们会加入其他的报表对象，例如 TQRLabel 对象、TQRDBText 对象及 TQRSysData 对象等，以增加报表的实用性。在报表程序制作上，我们至少会使用两个 TForm 对象，第一个 TForm 对象作为主画面，第两个 TForm 对象作为报表的设计。

范例 19-6

目的：本范例主要说明如何使用 Delphi 6 报表对象制作直列式(List)报表。

步骤 1 创建新的工程。

步骤 2 选择下拉式菜单“File/New Form”，为工程加入一个新的 TForm 对象，将其 Name 属性设为 Form2。

步骤 3 点选 Form1(工程默认的 TForm 对象)，并选择下拉式菜单“File/Use Unit”，且选择 Unit2，再单击“OK”按钮。

步骤 4 在 Form1 中加入两个 TButton 对象，其相关属性设置如表 19.13 所示。

表 19.13

对象	属性名称	属性设置值
Tbutton	Name	ButtonPrint
	Caption	打印
Tbutton	Name	ButtonPreview
	Caption	打印预览

步骤 5 在 ButtonPrint 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPrintClick(Sender: TObject);
begin
   //打印
   Form2.QuickRep1.Print;
end;
```

步骤 6 在 ButtonPreview 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonPreviewClick(Sender: TObject);
begin
   //打印预览
   Form2.QuickRep1.Preview;
end;
```

步骤 7 在 Form2 对象中加入一个 TTable 对象、一个 TQuery 及一个 TQuickRep 对象，其相关属性设置如表 19.14 所示。

表 19.14

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	EMPLOYEE.DB
	IndexFieldName	EmpNo
	Active	True

续上表

对象	属性名称	属性设置值
TQuery	Name	Query1
	SQL	Select Sum(Salary) form employee.db
	Active	True
TQuickRep	Name	QuickRep1
	DataSet	Table1
TQuickRep.Bands	HasColumnHeader	True
	HasDetail	True
	HasPageFooter	True
	HasPageHeader	True
	HasSummary	True
	HasTitle	True



如果您想要设置 TQuickRep 对象的其他属性，您除了可以通过对象检查器设置外，您也可以在 TQuickRep 对象上按右键，选择 Report Settings，以打开“Report Settings 对话框”。在此对话框中，您可以直接设置与报表版面有关的属性，例如纸张大小、边界及框线等，如图 19.22 所示。



图 19.22　TQuickRep 对象的版面设置

8 在 PageHeader Band 中加入一个 TQRLabel 对象及一个 TQRSysData 对象，其相关属性设置如表 19.15 所示。

表 19.15

对象	属性名称	属性设置值
TQRLabel	Name	QRLabelCoName
	Caption	卡洛斯工作室员工工资报表
	Font.Size	14

续上表

对象	属性名称	属性设置值
TQRSysData	Name	QRSysDataPrintTime
	Data	QrsDateTime

STEP 9 在 Title Band 中加入一个 TQRLabel 对象，其相关属性设置如表 19.16 所示。

表 19.16

对象	属性名称	属性设置值
TQRLabel	Name	QRLabelMonth
	Caption	十一月份工资
	Font.Size	14

STEP 10 在 ColumnHeader Band 中加入四个 TQRLabel 对象，其相关属性设置如表 19.17 所示。

表 19.17

对象	属性名称	属性设置值
TQRLabel	Name	QRLabelEmpNo
	Caption	员工编号
TQRLabel	Name	QRLabelLastName
	Caption	LastName
TQRLabel	Name	QRLabelFirstName
	Caption	FirstName
TQRLabel	Name	QRLabelSalary
	Caption	工资

STEP 11 在 Detail Band 中加入四个 TQRDBText 对象，相关属性设置如表 19.18 所示。

表 19.18

对象	属性名称	属性设置值
DetailBand1	Frame.DrawButton	True(画出边框)
	Frame.Left	True(画出边框)
	Frame.Right	True(画出边框)
	Frame.Top	True(画出边框)
TQRDBText	Name	TQRDBTextEmpNO
	DataSet	Table1
	DataField	EmpNo
TQRDBText	Name	TQRDBTextLastName
	DataSet	Table1
	DataField	LastName

续上表

对象	属性名称	属性设置值
TQRDBText	Name	TQRDBTextFirstName
	DataSet	Table1
	DataField	FirstName
TQRDBText	Name	TQRDBTextSalary
	DataSet	Table1
	DataField	Salary

Step 12 在 Summary Band 中加入一个 TQRLabel 对象与一个 TQRDBText 对象，其相关属性设置如表 19.19 所示。

表 19.19

对象	属性名称	属性设置值
TQRLabel	Name	QRLabelTotal
	Caption	总工资
TQRDBText	Name	QRDBTextTotal
	DataSet	Query1
	DataField	SUM OF Salary

Step 13 在 PageFooter Band 中加入两个 TQRLabel 对象及一个 TQRSysData 对象，其相关属性设置如表 19.20 所示。

表 19.20

对象	属性名称	属性设置值
TQRLabel	Name	QRLabelNum1
	Caption	第
TQRLabel	Name	QRLabelNum2
	Caption	页
TQRSysData	Name	QRSysDataPage
	Data	QrsPageNumber

Step 14 保存工程(对象的摆放位置如图 19.23 所示)。



图 19.23　报表设计画面

15 按【F9】功能键运行工程。工程运行后，您可单击“打印”按钮直接打印。也可以单击“打印预览”按钮预览所要打印的数据，如图 19.24 所示。



图 19.24　报表预览画面

注意　您也可以在程序运行中动态修改报表上的某个对象，这里以“月份标题”为例，您只要通过下面的语法就可以动态修改“月份标题”：Form2.QRLabelMonth.Caption := FormatDateTime ('yyyy"年"mm"月份工资表"', Now);

19.2.4　标签式(Label)报表制作

标签式(Label)报表通常是用于打印邮寄标签。标签式报表在制作上非常便利，只要您为 TQuickRep 对象添加一个 TDetailBand 对象，并将 TQuickRep.Page.Columns 属性值设为所需要的“列数(Column)”即可。

范例 19-7

目的：本范例主要说明如何使用 Delphi 6 报表对象制作标签式(Label)报表。

1 创建新的工程。

2 选择下拉式菜单“File/New Form”，为工程加入一个新的 TForm 对象，将其 Name 属性设为 Form2。

(3) 点选 Form1(工程默认的 TForm 对象)，并选择下拉式菜单“File/Use Unit”，且选择 Unit2，再单击“OK”按钮。

(4) 在 Form1 中加入两个 TButton 对象，其相关属性设置如表 19.21 所示。

表 19.21

对象	属性名称	属性设置值
Tbutton	Name	ButtonPrint
	Caption	打印
Tbutton	Name	ButtonPreview
	Caption	打印预览

(5) 在 ButtonPrint 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPrintClick(Sender: TObject);
begin
   //打印
   Form2.QuickRep1.Print;
end;
```

(6) 在 ButtonPreview 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonPreviewClick(Sender: TObject);
begin
  //打印预览
  Form2.QuickRep1.Preview;
end;
```

(7) 在 Form2 对象中加入一个 TTable 对象及一个 TQuickRep 对象，其相关属性设置如表 19.22 所示：

表 19.22

对象	属性名称	属性设置值
Ttable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	vendors.db
	Active	True
TquickRep	Name	QuickRep1
	DataSet	Table1
TQuickRep.Bands	HasDetail	True
TQuickRep.Page	Columns	3

(8) 在 TQuickRep 对象的 DetailBand 中加入两个 TQRDBText 对象，其相关属性设置如表 19.23 所示。

表 19.23

对象	属性名称	属性设置值
TQRDBText	Name	QRDBText1
	DataSet	Table1
	DataField	VendorName
	Font	(根据需要自行调整)
TQRDBText	Name	QRDBText2
	DataSet	Table1
	DataField	Address1
	Font	(根据需要自行调整)
	WordWrap	True
	AutoSize	False

(9) 为了让 QRDBText2 对象的数据能够顺利折行显示，因此我们特别将其 WordWrap 属性设为 True，并且将 AutoSize 设为 False。除此之外，您还必须指定 QRDBText2 对象的显示范围，以供换行显示之用，如图 19.25 所示。

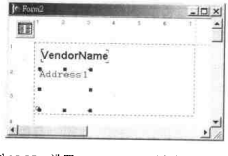


图 19.25　设置 TQRDBText 对象的显示范围

(10) 保存工程。

(11) 按【F9】功能键运行工程。工程运行后，您可单击“打印”按钮直接打印。也可以单击“打印预览”按钮预览所要打印的数据，如图 19.26 所示。

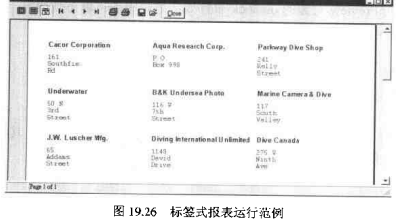
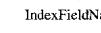


图 19.26　标签式报表运行范例

19.2.5 分组式(Group)报表制作

分组式(Group)报表也是一种经常被使用的报表类型。如果您是使用 TQRGroup 对象来设置分组(Group)条件时，您必须特别留意，所分组的数据字段最好与 TTable 对象的 IndexFieldNames 属性值或 IndexName 属性值相同。



19-8

目的：本范例主要说明如何使用 Delphi 6 报表对象制作分组式(Group)报表。

step 1 创建新的工程。

step 2 选择下拉式菜单“File/New Form”，为工程加入一个新的 TForm 对象，将其 Name 属性设为 Form2。

step 3 点选 Form1(工程默认的 TForm 对象)，并选择下拉式菜单“File/Use Unit”，且选择 Unit2，再单击“OK”按钮。

step 4 在 Form1 中加入两个 TButton 对象，其相关属性设置如表 19.24 所示。

表 19.24

对象	属性名称	属性设置值
Tbutton	Name	ButtonPrint
	Caption	打印
Tbutton	Name	ButtonPreview
	Caption	打印预览

step 5 在 ButtonPrint 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPrintClick(Sender: TObject);
begin
   //打印
   Form2.QuickRep1.Print;
end;
```

step 6 在 ButtonPreview 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonPreviewClick(Sender: TObject);
begin
   //打印预览
   Form2.QuickRep1.Preview;
end;
```

step 7 在 Form2 对象中加入一个 TTable 对象及一个 TQuickRep 对象，其相关属性设置如表 19.25 所示。

表 19.25

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	orders.db
	IndexFieldNames	CustNo
	Active	True
TQuickRep	Name	QuickRep1
	DataSet	Table1
TQuickRep.Bands	HasDetail	True

Step 8 在 TQuickRep 对象的 DetailBand 中加入三个 TQRDBText 对象，其相关属性设置如表 19.26 所示。

表 19.26

对象	属性名称	属性设置值
TQRDBText	Name	QRDBText1
	DataSet	Table1
	DataField	OrderNo
TQRDBText	Name	QRDBText2
	DataSet	Table1
	DataField	ItemsTotal
TQRDBText	Name	QRDBText3
	DataSet	Table1
	DataField	PaymentMethod

Step 9 在 TQuickRep 对象中加入一个 TQRGroup 对象。在 Expression 属性中所设置的条件即为分组(Group)的依据，您可以根据实际需要输入所要分组的条件，TQRGroup 对象的相关属性设置如表 19.27 所示。

表 19.27

对象	属性名称	属性设置值
TQRGroup	Name	QRGroup1
	Expression	Table1.CustNo

Step 10 在 TQRGroup 对象中加入一个 TQRExpr 对象。TQRExpr 对象的 Expression 属性值通常会与 TQRGroup 对象的 Expression 属性值相同。因为 TQRGroup 对象的 Expression 属性值是分组的依据，而 TQRExpr 对象的目的，只是将分组条件显示给用户而已。TQRExpr 对象的相关属性设置如表 19.28 所示。

表 19.28

对象	属性名称	属性设置值
TQRExpr	Name	QRExpr1
	Expression	Table1.CustNo

STEP 11 保存工程。

STEP 12 按【F9】功能键运行工程。工程运行后，您可单击“打印”按钮直接打印。也可以单击“预览打印”按钮预览所要打印的数据。如图 19.27 所示。

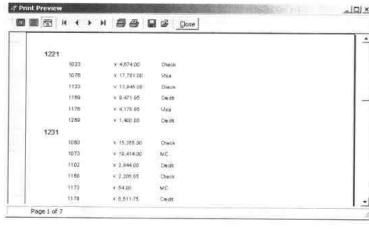


图 19.27 分组式报表运行范例

19.2.6 Master/Detail 报表制作

Master/Detail 报表在实务上也是一种经常被使用的报表格式，例如某个客户的购买清单、消费者的消费明细或刷卡明细等。在设计上，除了使用 TQuickRep 对象的 TDetailBand 对象外，我们会另外配合 TQRSubDetail 对象一起使用。关于 TDataSet 的设计就如同 Master/Detail 数据库应用程序一样，我们会将其设置为 Master/Detail 的关联关系，详细说明请参考第 17 章。

范例 19-9

目的：本范例主要说明如何使用 Delphi 6 报表对象制作 Master/ Detail 报表。

STEP 1 创建新的工程。

STEP 2 选择下拉式菜单“File/New Form”，为工程加入一个新的 TForm 对象，将其 Name 属性设为 Form2。

STEP 3 点选 Form1(工程默认的 TForm 对象)，并选择下拉式菜单“File/Use Unit”，且选择 Unit2，再单击“OK”按钮。

STEP 4 在 Form1 中加入两个 TButton 对象，其相关属性设置如表 19.29 所示。

表 19.29

对象	属性名称	属性设置值
TButton	Name	ButtonPrint
	Caption	打印
TButton	Name	ButtonPreview
	Caption	打印预览

5 在 ButtonPrint 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ButtonPrintClick(Sender: TObject);
begin
   //打印
   Form2.QuickRep1.Print;
end;
```

6 在 ButtonPreview 对象的 onClick 事件处理过程中加入以下代码：

```
procedure TForm1.ButtonPreviewClick(Sender: TObject);
begin
  //打印预览
  Form2.QuickRep1.Preview;
end;
```

7 在 Form2 对象中加入两个 TTable 对象、一个 TDataSource 对象及一个 TQuickRep 对象，其相关属性设置如表 19.30 所示。

表 19.30

对象	属性名称	属性设置值
TTable	Name	Table1
	DatabaseName	DBDEMOS
	TableName	customer.db
	IndexFieldNames	CustNo
	Active	True
TDataSource	Name	DataSource1
	DataSet	Table1
TTable	Name	Table2
	DatabaseName	DBDEMOS
	TableName	orders.db
	IndexName	CustNo
	MasterSource	DataSource1
	MasterFields	CustNo
	Active	True

续上表

对象	属性名称	属性设置值
TQuickRep	Name	QuickRep1
	DataSet	Table1
TQuickRep.Bands	HasDetail	True

步骤 8 在 TQuickRep 对象的 DetailBand 中加入三个 TQRDBText 对象及三个 TQRLabel 对象，其相关属性设置如表 19.31 所示。

表 19.31

对象	属性名称	属性设置值
TQRDBText	Name	QRDBText1
	DataSet	Table1
	DataField	CustNo
TQRDBText	Name	QRDBText2
	DataSet	Table1
	DataField	Company
TQRDBText	Name	QRDBText3
	DataSet	Table1
	DataField	Phone
TQRLabel	Name	QRLabel1
	Caption	OrderNo
TQRLabel	Name	QRLabel2
	Caption	ItemsTotal
TQRLabel	Name	QRLabel3
	Caption	PaymentMethod

步骤 9 在 TQuickRep 对象中加入一个 TQRSubDetail 对象，并在 TQRSubDetail 对象中加入三个 TQRDBText 对象，其相关属性设置如表 19.32 所示。

表 19.32

对象	属性名称	属性设置值
TQRSubDeatil	Name	QRSubDetail
	DataSet	Table2
	LinkBand	DetailBand
TQRDBText	Name	QRDBText4
	DataSet	Table2
	DataField	OrderNo

续上表

对象	属性名称	属性设置值
TQRDBText	Name	QRDBText5
	DataSet	Table2
	DataField	ItemsTotal
TQRDBText	Name	QRDBText6
	DataSet	Table2
	DataField	PaymentMethod

(10) TQRDBText 对象与 TQRLabel 对象的显示字体，请自行设计，这里不再说明。关于对象的摆放位置，如图 19.28 所示。

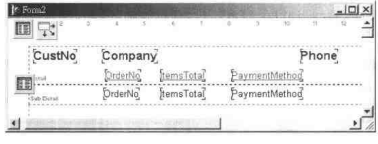


图 19.28　Master/Detail 报表对象摆放位置

(11) 保存工程。

(12) 按【F9】功能键运行工程，工程运行后，您可单击“打印”按钮直接打印。也可以单击“打印预览”按钮预览所要打印的数据，如图 19.29 所示。

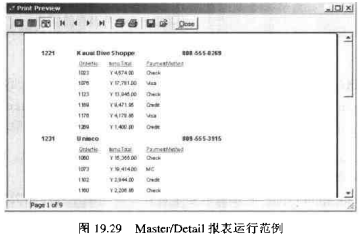


图 19.29　Master/Detail 报表运行范例

19.3　Delphi 6 的安装程序

您可以从 Delphi 6 原版光盘中安装 Delphi 6 所附的安装程序(Install Shield Express Borland

Limited Edition)。Install Shield 是所有安装软件中最常被使用的，在 Delphi 6 以前版本的光盘中您也可以找到 Install Shield。以下所要示范的是 3.03 版，这个版本要比之前所附的 Install Shield 还要好用，它可以轻易的包装应用程序中所需要的任何文件，包括运行文件、数据文件及每个组件所需的.dll 文件。您可以在原版光盘的 ISxpress 目录下找到 Setupex.exe，这就是 Delphi 6 所附的安装程序。

以下将以 Step by Step 的方式为大家说明如何使用 Install Shield Express，在这之前我们必须先创建一个简单的数据库应用程序及相关的说明文件(您可以在本书附赠的光盘中找到这个范例)，这个应用程序所使用的数据库别名为 myMSSQL，它连接至 MS Server 2000 数据库服务器。这些是非常重要的信息，因为我们将会在安装程序的制作过程中使用它们。

1 首先，请您选择“开始/程序/InstallShield/Express - Borland Limited Edition”，以打开“Install Shiled Express”。当您正确打开 Install Shield 后，系统会出现一个选择的画面(如图 19.30 所示)。在这个画面中，请先选择“Create a new project...”选项，并在“Project Name and Location”字段中决定此安装程序的保存位置与名称，最后再单击“Create”按钮。

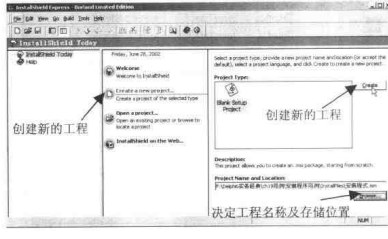


图 19.30 创建新的工程

注意

一个安装程序需要设置的数据包括：“Organize Your Setup”、“Specify Application Data”、“Configure the Target System”、“Customize the Setup Appearance”、“Define Setup Requirements and Actions”及“Prepare for Release”等六个主要步骤，分述如下。

- Organize Your Setup：此步骤主要是设置一些与组织相关的信息，例如：作者、安装目录、安装类型或连络方式等。
- Specify Application Data：此步骤主要是设置一些与应用程序相关的信息，例如：每一种安装方式所需要的文件及应用程序所需要的对象等，它也可以并入另一个已经存在的安装程序。
- Configure the Target System：此步骤主要是设置一些将来可能被安装的计算机的相关信息，例如：程序的目录或快捷方式的设置、系统注册文件的设置、ODBC 或 ini 文件的设置等。
- Customize the Setup Appearance：此步骤主要是决定安装过程所需要出现的画面。

- Define Setup Requirements and Actions：此步骤主要是决定安装程序运行前的判断工作，例如：操作系统种类、CPU 与 RAM 的最低需求限制及屏幕分辨率的限制等。
- Prepare for Release：此步骤主要在制作安装程序、运行测试及分发安装程序等。

(2) 请点选“Organize Your Setup”步骤中的“General Information”选项，请参考图 19.32 的示范说明。其中 Subject 代表此安装程序的标题名称；Product name 代表产生名称；INSTALLDIR 代表应用程序的安装目录；DATABASEDIR 代表数据库的安装目录；另外“Disable Change Button”、“Disable Remove Button”及“Disable Repair Button”等选项是用来决定在 Windows 2000 控制面板的添加/删除程序中是否出现“更改”或“删除”的按钮，如图 19.31 所示。

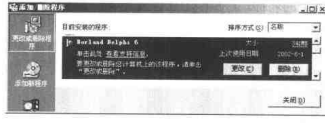


图 19.31　Windows 2000 的添加/删除程序

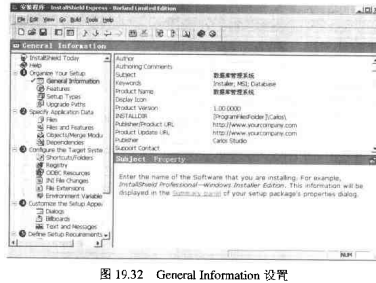


图 19.32　General Information 设置

(3) 请点选“Organize Your Setup”步骤中的“Features”选项，请参考图 19.33 的示范说明。所谓的“Features”是将需要被安装的文件按照其功能加以分类，例如:可执行文件、帮助文件或其他参考的文件等。您只要在“Features”选项上按右键并选择“New Feature Ins”即可加入新的分类名称，这里以“My Feature 1”及“My Feature 2”为例。

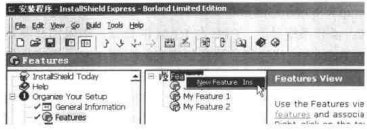


图 19.33　将所有被安装的程序加以分类

4 请点选“Organize Your Setup”步骤中的“Setup Types”选项，请参考图 19.34 的示范说明。系统提供“Typical”、“Minimal”及“Custom”等安装类型，您可以自行修改这些类型的名称。这里除了让您设置安装类型名称外，主要是让您决定每一种安装类型所需要的 Feature。我们这里将 Typical 安装类型的内容设置为“Always Install”及“My Feature 1”等；Minimal 安装类型的内容只包括“Always Install”而已；而 Custom 安装类型的内容则包括“Always Install”、“My Feature 1”及“My Feature 2”等。

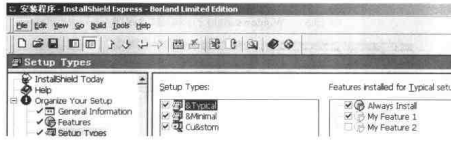


图 19.34　决定每一个安装类型所需要的 Feature

5 请点选“Specify Application Data”步骤中的“Files”选项，请参考图 19.35 的示范说明。这个步骤主要的工作是为每一个 Feature 指定其所需要的文件。这里为“Always Install”选项加入 Project1.exe；为“My Feature 1”选项加入 Readme.txt；为“My Feature 2”选项加入 Readme.htm 及 Readme.files 目录。



图 19.35　决定每一个 Feature 所需要的文件

注意

在 Files 选项中，除了可以加入应用程序必要的文件外，您也可以将数据库的驱动程序一并加入此安装程序中。图 19.36 以加入 MS Server 2000 的 NTWDBLIB.DLL 驱动程序为例。

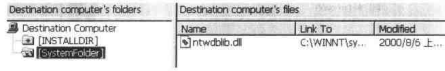


图 19.36　加入数据库驱动程序

6 请点选“Specify Application Data”步骤中的“Files and Feature”选项，请参考图 19.37 的示范说明。这个选项主要是让您再一次检查所有 Feature 所包含的文件名称是否正确，并允许做适当的修改。

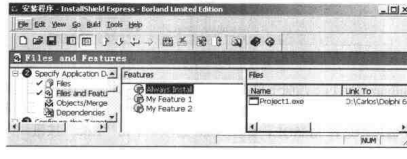


图 19.37　查看每一个 Feature 所需要的文件

7 请点选“Specify Application Data”步骤中的“Objects/Merge Modules”选项，请参考图 19.38 的示范说明。这个选项主要是让您加入所需的各种组件，包括：ADO、DCOM 95 及 QuickReport 等。它也可以允许您附带安装其他的安装程序如 ACCESS 97 或 ACCESS 2000 等。因为这是一个使用 BDE 的数据库应用程序，因此，笔者选择“BDERTL”及“BDE_ENT”等。此时系统自动将“BaseRTL”、“DatabaseRTL”等选项自动加入。当您点选“BDE_ENT”选项时，系统会自动打开“BDE Object Wizard”，如图 19.18 所示的画面。



图 19.38　BDE Object Wizard

8 在 BDE Object Wizard 的欢迎画面后，请指定 BDE 的 INI 设置文件的保存位置，并单击“下一步”按钮，如图 19.39 所示。

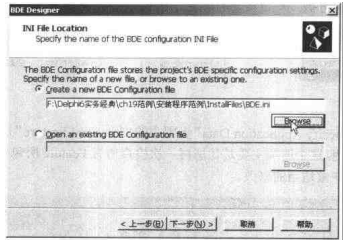


图 19.39 BDE Object Wizard

9 在 Configure INI File 画面中单击“Launch”按钮，以进行 BDE 别名的设置。此时系统出现一个“BDE Merge Module Configuration Utility”对话框，请在此对话框的 Driver 选项中选择 MSSQL，并在 Alias 选项中单击“Add”按钮。

10 此时系统出现另一个“Add alias”对话框，请在此对话框的 Alias name 字段中选择“myMSSQL”。当您选择指定的别名后，系统会带出相关的信息，如 Driver Name 或相关的参数设置值，如图 19.40 所示

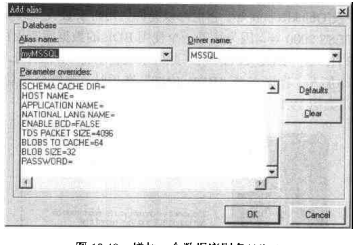


图 19.40 增加一个数据库别名(Alias)

11 指定数据库别名后，“BDE Merge Module Configuration Utility”对话框的内容如图 19.41 所示。当您确认数据无误后，请保存并关闭此对话框，并结束 BDE Object Wizard。

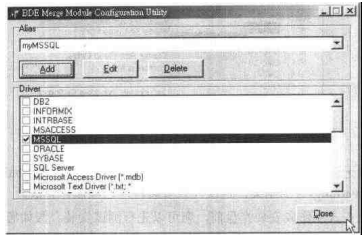


图 19.41　BDE Merge Module Configuration Utility

(12) 请点选“Configure the Target System”步骤中的“Shortcuts /Folders”选项，请参考图 19.42 的示范说明。这个选项主要是让您设置程序、启动、Send to 或是桌面上的数据项。这里示范在程序集中创建一个“我的数据库应用程序”目录，并为此目录加入“程序”、“HTM 帮助文档”及“文本文件帮助文档”等快捷方式选项。您可以通过“Feature”与“Target”等字段决定所对应的文件名称。



图 19.42　设置程序集的目录与快捷方式

(13) 在“Configure the Target System”步骤中的“Registry”、“ODBC Resources”、“INI File Changes”及“File Extensions”选项在本范例中并没有作用。其分别代表的意义为：Registry 代表系统注册文件的内容、ODBC Resources 代表 ODBC 的 DSN、INI File Changes 代表使用 ini 为扩展名的设置文件、而 File Extensions 则代表为指定的扩展名设置打开的可执行文件。

(14) 在“Customize the Setup Appearance”步骤中的“Dialogs”选项允许您设置安装过程中所要出现的画面。

(15) 在“Define Setup Requirements and Actions”步骤中的“Requirements”选项中允许您设置安装时的必要检查选项，包括：操作系统版本、CPU 等级限制、内存限制、屏幕分辨率或颜色的限制等，本范例将以默认值为例。

(16) 接着，您可以通过“Prepare for Release”步骤中的“Build Your Release”选项来制作完整的打包程序，如图 19.43 所示。

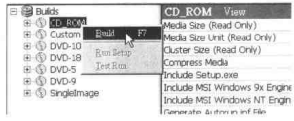


图 19.43　制作安装程序

17 如果设置无误，安装程序最后应该会出现“Express\CD_ROM build completed with 0 errors, 0 warnings”的结果，这就代表安装程序已经制作完成。

18 在您尚未正式制作成安装光盘前，您可以先行测试安装。假如您要测试安装，只要点选“Run Setup”即可(如图 19.45 所示)，测试安装的画面如图 19.44 所示。

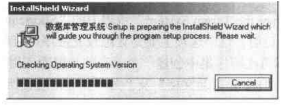


图 19.44　测试安装

19 安装完成后，您可以从“开始/程序”中找到“我的数据库应用程序”。如果您是选择 Customer 安装类型，那么您可以在该目录下找到“HTM 帮助文档”、“文本文件帮助文档”及“程序”等，如图 19.45 所示。这些选项运行画面分别如图 19.46、图 19.47 及图 19.48 所示。



图 19.45　安装完成

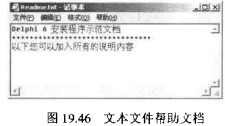


图 19.46　文本文件帮助文档



图 19.47　应用程序运行结果



图 19.48　HTML 帮助文档

步骤 20 最后，您可以通过 Distribute Your Release 功能进行安装程序的分发工作，如图 19.49 所示。

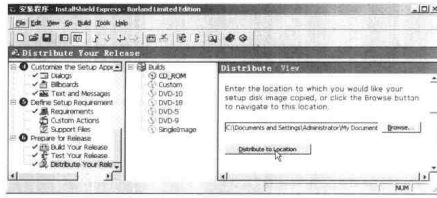


图 19.49　分发安装程序

第 20 章

Delphi 6 与 ADO 组件

本章目标

ADO(ActiveX Data Objects)最初是由 Microsoft 所制定的，它是一组能提供应用程序通过 OLE DB Provider 来访问数据库的组件。在 ASP(Active Server Pages)中就是利用 ADO 通过 ODBC(Open Database Connectivity)来访问各种数据库。而 Delphi 6 所提供的 ADO 组件则是将整个 Microsoft ADO 组件加以封装并以 VCL 组件呈现，通过 ADO 组件您所开发的数据库应用程序完全不需要依赖 BDE(Borland Database Engine)。

本章主题

20.1 何谓 ADO(ActiveX Data Objects)组件

20.2 TADOConnection 对象

20.3 TADODataSet 对象

20.4 TADOTable 对象

20.5 TADOQuery 对象

20.6 TADOStoredProc 对象

20.7 TADOCommand 对象

20.1 何谓 ADO(ActiveX Data Objects)组件

ADO(ActiveX Data Objects)最初是由 Microsoft 所制定的，它是一组能提供应用程序通过 OLE DB Provider 来访问数据库的组件。在 ASP(Active Server Pages)中就是利用 ADO 通过 ODBC(Open Database Connectivity)来访问各种数据库。而 Delphi 6 所提供的 ADO 组件则是将整个 Microsoft ADO 组件加以封装，通过 ADO 组件您可以完全不需要依赖 BDE 便能开发出功能完整的数据库应用程序。

ADO(ActiveX Data Objects)是微软公司提供网页或数据库应用程序开发者在网页或应用程序中访问数据库的最新技术，微软公司的 ADO 主要是由 Connection 对象、Command 对象、Parameter 对象、RecordSet 对象、Field 对象、Property 对象及 Error 对象等七个对象与 Fields 数据集、Properties 数据集、Parameters 数据集及 Errors 数据集等四个数据集所组合而成，其功能概述如下。ADO 提供的七个对象与四个数据集分述如下：

Connection 对象负责与指定的数据源进行连接，Connection 对象除了与数据源进行连接之外，它还可以通过事务(Transaction)来确保在事务中所有数据源变更的全部成功。以银行转帐操作而言，系统只允许两个指定帐号的金额必须成功的增减，否则视为事务失败。因此当事务失败时必须将指定帐号恢复至事务的最初。

1. Command 对象负责对数据库提供请求，也就是传递指定的 SQL 命令。换言之，Command 对象必须通过一个已经创建的连接(Connection 对象)发出数据处理语言(DML，Data Manipulation Language)来操作数据源的数据，这些命令通常包括 INSERT(新增记录)、DELETE(删除记录)、UPDATE(更新记录)或是 SELECT(以 Recordset 的形式捕获数据)等。除此之外，您也可以通过 Command 对象对数据库发出数据定义语言(DDL，Data Definition Language)，例如 CREATE TABLE(创建数据表)、DROP TABLE(删除数据表)或 ALTER TABLE(修改数据表结构)等。
2. Parameter 对象负责传递 Command 对象所需要的 SQL 命令参数。
3. RecordSet 对象负责浏览与操作从数据库取出的数据，换言之，Recordset 对象就像是一个二维的数组，数组的每一行代表一个数据行，而每个数据行包含一个或多个数据字段，这里的一个数据字段就是代表一个 Field 对象。
4. Field 对象代表指定 RecordSet 对象的数据字段。
5. Property 对象代表 ADO 的各项对象属性值，换言之，每一个 ADO 对象都会有一组描述或控制对象行的属性。
6. Error 对象负责记录连接过程所发生的错误信息。
7. Connection 对象包含 Errors 数据集，在 Errors 数据集中包含数据源响应单一失败所创建的 Error 对象。
8. Command 对象包含 Parameters 数据集，在 Parameters 数据集中包括 Command 对象所有的 Parameter 对象。
9. Recordset 对象包含 Fields 数据集，在 Fields 数据集中包含 Recordset 对象的所有 Field 数据字段对象。
10. Connection 对象、Command 对象、Recordset 对象与 Field 对象都包含一个 Properties

数据集，在 Properties 数据集中包含所对应 Connection 对象、Command 对象、Recordset 对象与 Field 对象的 Property 对象。

11. 在您了解 Microsoft ADO 所提供七个对象与四个数据集的功能后，我们将利用图标的方式来说明 Connection 对象、Command 对象、Parameter 对象、Recordset 对象、Field 对象、Property 对象及 Error 对象间的关系，如图 20.1 所示。

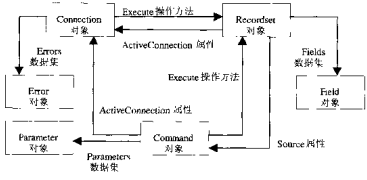


图 20.1　Microsoft ActiveX Data Object 关系图

Delphi 6 所提供的 ADO 数据库访问组件完全根据 Microsoft ADO 组件加以封装，通过 ADO 组件您可以完全不需要依赖 BDE 便能开发出功能完整的数据库应用程序。但是客户端计算机中必须安装 ADO 2.1 或是更新的版本，而且还是必须安装数据库系统的客户端软件，或是专属的 OLE DB 驱动程序或是专属的 ODBC 驱动程序。虽然免不了还是要安装数据库相关的驱动程序，但总比依赖 BDE 要来得好。就笔者的观点，应尽早将传统的数据库程序改为多层分布式应用程序，否则写数据库应用程序永远不会是一件愉快的事。

Delphi 6 原版光盘中所提供的 ADO 版本为 2.5。该安装程序的名称为 mdac_typ.exe，您可以从原版光盘的“Mdac”目录中找到它。在微软较新的操作系统(如 Windows ME 或 Windows 2000 等)中都已经安装了 ADO 组件，因此您无需为客户端的计算机另外安装。

因为 Delphi 6 的 ADO 数据库组件完全封装自 Microsoft ADO 组件，因此大部分的组件都可以对应于图 20.1。Connection、Command 及 Recordset 组件为 Microsoft ADO 的主要组件，其对应于 TADOConnection、TADOCommand 及 TADODataSet 对象。至于其他的 Parameter、Error 及 Field 对象等，都已经封装于对应的组件中，以下就 Delphi 6 所提供的 ADO 数据库组件加以简要说明。

1. TADOConnection 对象与 BDE 的 TDatabase 对象在功能上是对应的。其负责与数据库获取连接，并可以供其他 TADODataSet、TADOTable、TADOQuery、TADOStoredProc 与 TADOCommand 对象共享连接(session)，通过单一的连接来获取数据或运行命令等。更详细的介绍请参考第 20.2 节。
2. TADODataSet 对象并没有直接对应到 BDE 的 VCL 组件，其功能与 TTable 对象或 TQuery 对象较为接近，但其可从一个或多个表(Table)中获取数据。在实务上 TADOConnection 对象负责获取连接，而 TADODataSet 对象负责获取与操控数据。

更详细的介绍请参考第 20.3 节。

3. TADOTable 对象与 BDE 的 TTable 对象在功能上是对应的，其负责提供对单一表的相关操作。在实务上我们同样会使用 TADOConnection 对象负责获取连接。更详细的介绍请参考第 20.4 节。
4. TADOQuery 对象与 BDE 的 TQuery 对象在功能上是对应的，其负责通过 SQL 语句从数据库中获取数据。在实务上我们同样会使用 TADOConnection 对象负责获取连接。更详细的介绍请参考第 20.5 节。
5. TADOStoredProc 对象与 BDE 的 TStoredProc 对象在功能上是对应的，其负责通过运行数据库中的存储过程(Stored Procedure)以获取数据。在实务上我们同样会使用 TADOConnection 对象负责获取连接。更详细的介绍请参考第 20.6 节。
6. TADOCommand 对象并没有直接对应到 BDE 的 VCL 组件，负责运行 SQL 语句，但这些 SQL 语句通常不会返回数据集(DataSet)。换言之，如果需要运行返回数据集的 SQL 语句，我们通常会使用 TADOQuery 对象，但这并不代表 TADOCommand 对象无法运行返回数据集的 SQL 语句。在实务上我们同样会使用 TADOConnection 对象负责获取连接。更详细的介绍请参考第 20.7 节。

20.2 TADOConnection 对象

TADOConnection 对象与 BDE 的 TDatabase 对象在功能上是对应的。其负责与数据库获取连接，并可以供其他 TADODataSet、TADOTable、TADOQuery、TADOStoredProc 与 TADOCommand 对象共享连接(session)，通过单一的连接来获取数据或运行命令等。

20.2.1 TADOConnection 对象常用属性

以下就 TADOConnection 对象各种常用的属性加以说明之。

ConnectionString 属性

TADOConnection 对象的 ConnectionString 属性是一串用来连接数据库的必要字符串，它们包括用户名称、密码、远程数据库服务器名称或数据表所在的完整路径等，这些信息会以“;”来分隔并加以组合。但事实上，我们很少直接输入这些字符串，而是通过“连接信息编辑器”以步骤向导来完成。笔者将依次示范如何使用 TADOConnection 对象连接到 Access 数据库、Oracle 数据库与 MS SQL Server。以下先示范如何连接 Access 数据库，这里以 Delphi 6 默认的 Access 数据库为例，其保存在“C:\Program Files\Common Files\Borland Shared\Data\dbdemos.mdb”。



20-1

目的：本范例的目的是说明如何使用 TADOConnection 对象来创建与 Access 数据库的连接。

1 创建新的工程。

(2) 在 TForm 对象中加入一个 TADOConnection 对象，点选 TADOConnection 对象 ConnectionString 属性旁的按钮，以打开“连接信息编辑器”。并选择“Use Connection String”选项，再单击“Build”按钮，如图 20.2 所示。

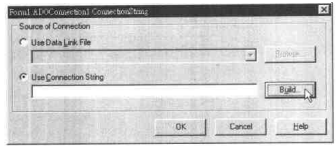


图 20.2　打开连接信息编辑器

(3) 此时连接向导出现一个“Data Link Properties”对话框，请从中选择合适的 OLE DB 驱动程序。因为我们要连接 Access 数据库，因此请您选择 Microsoft Jet OLEDB 4.0 并单击“Next”按钮，如图 20.3 所示。

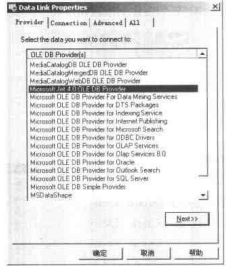


图 20.3　选择适合的 OLE DB 驱动程序

(4) 请在“Connection”页标签中指定您所要使用的 Access 数据库名称与用户帐号与密码。这里的数据库名称请使用“C:\ ProgramFiles\Common Files\BorlandShared\Data\dbdemos. mdb”，使用帐号请输入 Admin 而密码使用空白。这里的用户帐号与密码必须以 Access 数据库本身的设置为主，笔者是使用数据库的默认值，如图 20.4 所示。

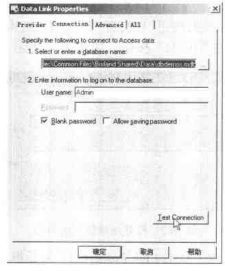


图 20.4　设置连接的相关信息

5　在设置连接信息后，您可以通过“Test Connection”按钮进行测试。如果连接信息设置无误，那么系统会响应“Test connection succeeded”的信息，如图 20.5 所示。



图 20.5　连接测试结果

6　您也可以从“Advanced”页标签中设置用户的访问权限。系统提供读取(Read)、读写(Read Write)、不拒绝共享(Share Deny None)、拒绝共享的读取(Share Deny Write)、拒绝共享的写入(Share Deny Read)、共享独占(Share Exclusive)与写入(Write)等访问权限，您可以同时指定一个或多个权限，关于访问权限的设置请参考 TADOConnection 对象的 Mode 属性相关说明，如图 20.6 所示。



图 20.6　设置访问权限

7　最后请单击“Data Link Properties”对话框中的“确定”按钮，再单击“连接信息编辑器”中的“OK”按钮以结束 TADOConnection 对象的 ConnectionString 属性设置。

步骤 8　当完成连接信息的设置后，系统会根据您的设置而自动更改相关的属性值，例如 Provider 属性、Mode 属性与 LoginPrompt 属性等，这些属性本小节稍后再另外说明。此时如果您将 Connected 属性设为 True，那么 TADOConnection 对象就会真正的与数据库获取连接。

步骤 9　TADOConnection 对象只负责与数据库获取连接，要从数据库获取数据或要通过其他的数据集(DataSet)对象。因此请您在 TForm 对象中加入一个 TADOTable 对象，并将其 Connection 属性设为 ADOConnection1。再从 Table 属性中选择适当的数据库表(Table)，这里请选择 employee 表。最后只要再将 Active 属性设为 True，TADOTable 对象便可以通过 TADOConnection 对象从数据库获取所指定的数据。

步骤 10　与开发 BDE 数据库应用程序相同，数据集(DataSet)对象必须通过 TDataSource 对象并配合其他数据库显示对象才能显示数据。常用的数据库显示对象，例如 TDBGrid 对象、TDBEdit 对象及 TDBText 对象等。因此请您在 TForm 对象中加入一个 TDataSource 对象、TDBGrid 对象与 TDBNavigator 对象，其相关属性设置如表 20.1 所示。

表 20.1

对象	属性名称	属性设置值
TDataSource	Name	DataSource1
	DataSet	ADOTable1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1

步骤 11　保存工程并按下【F9】功能键运行工程。工程运行后，您可以利用 TDBNavigator 对象来移动数据指针或是数据变动控制，如图 20.7 所示。

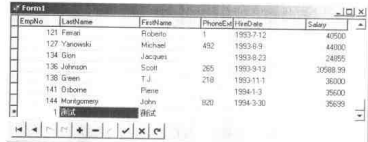


图 20.7　以 ADO 对象访问 Access 数据库运行结果

范例 20-2

目的：本范例的目的是说明如何使用 TADOConnection 对象来创建与 Oracle 8i 数据库的连接。

步骤 1　创建新的工程。

(2) 在 TForm 对象中加入一个 TADOConnection 对象，点选 TADOConnection 对象 ConnectionString 属性旁的按钮，以打开“连接信息编辑器”。并选择“Use Connection String”选项，再单击“Build”按钮，如图 20.8 所示。

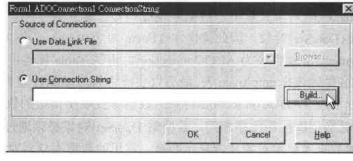


图 20.8 打开连接信息编辑器

(3) 此时连接向导出现一个“Data Link Properties”对话框，请从中选择合适的 OLE DB 驱动程序。因为我们要连接 Oracle 数据库，因此请您选择“Microsoft OLE DB Provider for Oracle”，并单击“Next”按钮，如图 20.9 所示。

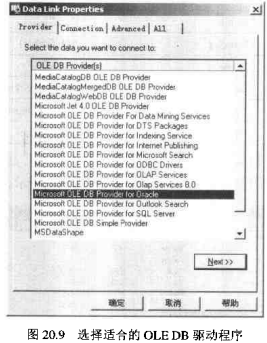


图 20.9 选择适合的 OLE DB 驱动程序

(4) 请在“Connection”页标签中指定您所要连接的 Oracle 数据库名称、用户帐号与密码。这里的用户帐号与密码以 scott 及 tiger 为例，如图 20.10 所示。

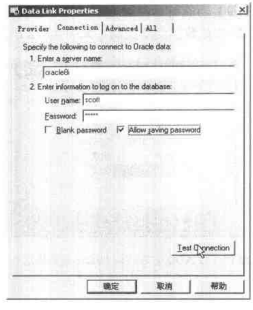


图 20.10　设置连接的相关信息

5 在设置连接信息后，您可以通过“Test Connection”按钮来进行测试。如果连接信息设置无误，那么系统会响应“Test connection succeeded”的信息，如图 20.11 所示。



图 20.11　连接测试结果

6 最后请单击“Data Link Properties”对话框中的“确定”按钮，再单击“连接信息编辑器”中的“OK”按钮以结束 TADOConnection 对象的 ConnectionString 属性设置。

7 当完成连接信息的设置后，系统会根据您的设置而自动更改相关的属性值，请参考范例 20-1 的详细说明。

8 请您在 TForm 对象中加入一个 TDataSource 对象、TDBGrid 对象与 TDBNavigator 对象，其相关属性设置如表 20.2 所示。

表 20.2

对象	属性名称	属性设置值
TDataSource	Name	DataSource1
	DataSet	ADOTable1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1

9 保存工程并按下【F9】功能键运行工程。工程运行后，您可以利用 TDBNavigator 对象来移动数据指针或是数据变动控制，如图 20.12 所示。



图 20.12 以 ADO 对象访问 Oracle 数据库运行结果

范例 20-13

目的：本范例的目的是说明如何使用 TADOConnection 对象来创建与 MS Server 2000 数据库的连接。

1 创建新的工程。

2 在 TForm 对象中加入一个 TADOConnection 对象，点选 TADOConnection 对象 ConnectionString 属性旁的按钮，以打开“连接信息编辑器”。并选择“Use Connection String”选项，再单击“Build”按钮，如图 20.13 所示。

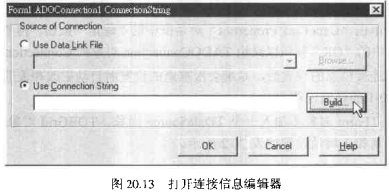


图 20.13 打开连接信息编辑器

3 此时连接向导出现一个“Data Link Properties”对话框，请从中选择合适的 OLE DB 驱动程序。因为我们要连接 MS SQL Server 数据库，因此请您选择“Microsoft OLE DB Provider for SQL Server”并单击“Next”按钮，如图 20.14 所示。

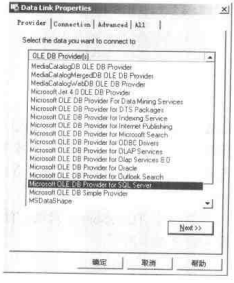


图 20.14　选择适合的 OLE DB 驱动程序

步骤 4　请在“Connection”页标签中指定您所要连接的 MS Server 数据库名称、用户帐号与密码。这里的用户帐号与密码以 sa 帐号为例，如图 20.15 所示。

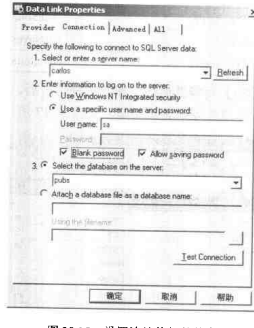


图 20.15　设置连接的相关信息

步骤 5　在设置连接信息后，您可以通过“Test Connection”按钮来进行测试。如果连接信息设置无误，那么系统会响应“Test connection succeeded”的信息，如图 20.16 所示。

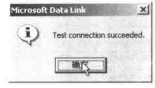


图 20.16 连接测试结果

6 最后请单击“Data Link Properties”对话框中的“确定”按钮，再单击“连接信息编辑器”中的“OK”按钮以结束 TADOConnection 对象的 ConnectionString 属性设置。

7 当完成连接信息的设置后，系统会根据您的设置而自动更改相关的属性值，请参考范例 20-1 的详细说明。

8 请您在 TForm 对象中加入一个 TDataSource 对象、TDBGrid 对象与 TDBNavigator 对象，其相关属性设置如表 20.3 所示。

表 20.3

对象	属性名称	属性设置值
TDataSource	Name	DataSource1
	DataSet	ADOTable1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1

9 保存工程并按下【F9】功能键运行工程。工程运行后，您可以利用 TDBNavigator 对象来移动数据指针或是数据更新控制，如图 20.17 所示。

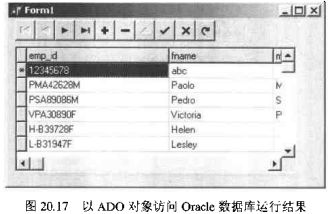


图 20.17 以 ADO 对象访问 Oracle 数据库运行结果

Connected 属性

TADOConnection 对象的 Connected 属性代表与数据库的连接状态，当 Connected 属性值为 True，表示已经与数据库获取连接。

Provider 属性

TADOConnection 对象的 Provider 属性代表连接时所使用的 OLEDB 驱动程序，Provider 属性值会根据 ConnectionString 属性中所指定的 Provider 而改变。

KeepConnection 属性

TADOConnection 对象的 KeepConnection 属性代表是否与数据库保持连接状态。当 KeepConnection 属性值为 True，表示就算没有任何一个 DataSet 对象与数据库连接，TADOConnection 对象也保持与数据库的连接状态。

LoginPrompt 属性

当 TADOConnection 对象的 LoginPrompt 属性值为 True，表示会出现数据库登录的对话框；反之，不会出现。如果您要将 LoginPrompt 属性值设为 False，那么您必须在 ConnectionString 属性中指定用户帐号与密码等信息，否则无法正确登录数据库。

CursorLocation 属性

TADOConnection 对象的 CursorLocation 属性决定与数据库连接时所使用的光标(cursor)类型。CursorLocation 属性包含客户端光标(clUseClient)及服务端光标(clUseServer)两种值。客户端光标是系统默认值，其作法是将获取的数据保存于客户端的计算机，因此它允许程序进行一些较为特殊的数据处理操作，例如重新排序(resorting)或数据过滤(filtering)等。相对于客户端光标，服务端光标是一种比较没有灵活性的光标类型，它并没有将所有数据保存于客户端的计算机中，因此不允许程序进行一些较为特殊的数据处理操作。但是当您处理大量的查询结果时，您必须使用服务端光标来解决大量数据传输的问题。

Mode 属性

TADOConnection 对象的 Mode 属性，其包含 cmUnknown、cmRead、cmWrite、cmReadWrite、cmShareDenyRead、cmShareDenyWrite、cmShareExclusive 及 cmShareDenyNone 等可能值。CmUnknown 代表尚未确定类型；读取(cmRead)代表以只读的类型打开数据库；写入(cmWrite) 代表以只写的类型打开数据库；读写(cmReadWrite)代表此数据库可供读取和写入；拒绝共享的写入(cmShareDenyRead) 代表数据库可以拒绝其他用户的写入；拒绝共享的读取(cmShareDenyWrite)代表数据库可以拒绝其他用户的读取；共享独占(cmShareExclusive)代表数据库会拒绝其他用户的读取或写入；不拒绝共享(cmShareDenyNone)代表数据库不可以拒绝其他用户的读取或写入。

State 属性

TADOConnection 对象的 State 属性是一个只读属性继承自 TObjectStates 对象，其包含 stClosed、stOpen、stConnecting、stExecuting 及 stFetching 等可能值。当 State 属性值为 stClosed 时则表示此 TADOConnection 对象是处于未连接(断开)状态。当 State 属性值为 stOpen 时则表示此 TADOConnection 对象是处于连接状态。当 State 属性值为 stConnecting 时则表示此

TADOConnection 对象处于尝试连接状态。当 State 属性值为 stExecuting 时则表示此 TADOConnection 对象处于运行状态。当 State 属性值为 stFetching 时，表示此 TADOConnection 对象处于获取数据的状态。

Attributes 属性

Attributes 属性用来控制 TADOConnection 对象是否保留 commits 与保留 aborts 等，其属性可能值包含 xaCommitRetaining 与 xaAbortRetaining 等。这里所谓的 Commit 是指确认事务(Committing a transaction)，而 Abort 是指取消事务(Rolling back a transaction)。

ConnectionTimeout 属性

ConnectionTimeOut 属性用来设置 TADOConnection 对象最长的连接等待时间，其默认值为 15 秒。当您调用 Open 操作方法或其他方式创建连接时，TADOConnection 对象会尝试与数据库获取连接，当连接的时间大于 ConnectionTimeout 属性值，那么系统会停止尝试连接。

ConnectionOptions 属性

ConnectionOptions 属性用来设置 TADOConnection 对象与数据库的连接模式，其包含 coConnectUnspecified 与 coAsyncConnect 等。前者代表由数据库服务器来决定最适合的连接模式，后者采用异步的连接模式。

CommandTimeOut 属性

CommandTimeOut 属性用来设置 TADOConnection 对象最长的命令运行等待时间，其默认值为 30 秒。当您调用 Execute 操作方法或其他方式运行命令时，假如 TADOConnection 对象运行命令的时间大于 CommandTimeout 属性值，那么系统会停止运行命令。

Commands 属性

通过 Commands 属性可以获取与 TADOConnection 对象连接的所有 TADOCommand 对象。

CommandCount 属性

通过 CommandCount 属性可以获取与 TADOConnection 对象连接的 TADOCommand 对象个数。

DataSets 属性

通过 DataSets 属性可以获取与 TADOConnection 对象连接的所有 DataSet 对象，包括 TADODataSet 对象、TADOTable 对象、TADOQuery 对象及 TADOStoredProc 对象等。

DataSetCount 属性

通过 DataSetCount 属性可以获取与 TADOConnection 对象连接的所有 DataSet 对象个数。这里所谓的 DataSet 对象包括 TADODataSet 对象、TADOTable 对象、TADOQuery 对象及 TADOStoredProc 对象等。

- *InTransaction 属性*

InTransaction 属性可以判断 TADOConnection 对象是否处于事务(Transaction)处理中，关于事务的实现请参考 BeginTrans、CommitTrans 及 RoolbackTrans 等操作方法。

- *Version 属性*

Version 属性可以获取目前 TADOConnection 对象的版本号码。

20.2.2　TADOConnection 对象常用操作方法

以下就 TADOConnection 对象各种常用的操作方法加以说明之。

- *BeginTrans 操作方法*

TADOConnection 对象的 BeginTrans 操作方法可以打开一个事务(Transaction)，这里所谓的事务是数据库一个相当重要的名称。事务的观念允许写同数据库的数据可以有恢复的机会，在使用 Commit 命令确认之前，都可以使用 RollBack 命令加以恢复。

- *RollbackTrans 操作方法*

TADOConnection 对象的 RollbackTrans 操作方法如同使用数据库的 Rollback 命令一样，可以用来取消事务。RollbackTrans 操作方法会将数据库的内容恢复到上一个 Commit 命令之后。

- *CommitTrans 操作方法*

ADOConnection 对象的 CommitTrans 操作方法就如同使用数据库的 Commit 命令一样，可以用来确认事务。CommitTrans 操作方法会确认(Commit)数据库的修改内容。在调用 CommitTrans 操作方法之后，无法再使用 RollbackTrans 操作方法恢复修改的数据。

- *Cancel 操作方法*

当 TADOConnection 对象处于尝试连接状态，其 State 属性值为 stConnecting。这时候您可以调用 TADOConnection 对象的 Cancel 操作方法来取消尝试连接的操作。

- *CloseDataSets 操作方法*

CloseDataSets 操作方法可以取消所有与 TADOConnection 对象连接的 DataSet 对象。此操作方法有一个参数值(all)，当参数值为 True，表示取消所有 DataSet 对象与数据库的连接。当参数值为 False，则表示关闭所有 CursorLocation 属性不为 clUseClient 的 DataSet 对象与数据库的连接。

- *GetProcedureNames 操作方法*

TADOConnection 对象的 GetProcedureNames 操作方法可以获取用户所有可以使用的存储

过程(Stored Procedure)名称。以下程序片段示范如何获取所有的存储过程名称，并将其保存到 TListBox 对象中。

```
ADOConnection1.GetProcedureNames(ListBox1.Items);
```

GetTableNames 操作方法

TADOConnection 对象的 GetTableNames 操作方法可以获取用户所有可以使用的数据库表(Table)名称。以下程序片段示范如何获取所有的数据表名称，并将其保存到 TListBox 对象中。

```
ADOConnection1.GetTableNames(ListBox1.Items, False);
```

范例 20-4

目的：本范例的目的是说明如何使用 TADOConnection 对象的 GetTableNames 操作方法获取用户所有可以使用的数据库表(Table)名称并显示该数据库表的内容。

1. 创建新工程。
2. 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 20-2 的 Oracle Server 连接示范。
3. 在 TForm 对象中另外加入一个 TADOTable 对象、一个 TDataSource 对象、TDBGrid 对象、TDBNavigator 对象、一个 TListBox 对象及两个 TLabel 对象，其相关属性设置如表 20.4 所示。

表 20.4

对象	属性名称	属性设置值
TADOTable	Name	ADOTable1
	Connection	ADOConnection1
TDataSource	Name	DataSource1
	DataSet	ADOTable1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1
TListBox	Name	ListBox1
TLabel	Name	Label1
	Caption	数据库表名称
TLabel	Name	Label2
	Caption	表内容

4. 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //获取数据库中所有的数据表
  ADOConnection1.GetTableNames(ListBox1.Items, False);
end;
```

⑤ 在 ListBox1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
   //设置被显示的数据表名称
   with ADOTable1 do
       begin
          Active := False;
          TableName := ListBox1.Items[ListBox1.ItemIndex];
          Active := True;
       end;
end;
```

⑥ 保存工程，并按下【F9】运行工程。工程运行后，系统会加载相关的数据库表，只要您点选 TListBox 中的表名称，则该表的内容将会被显示在 TDBGrid 对象中，如图 20.18 所示。



图 20.18　GetTableNames 操作方法运行范例

Execute 操作方法

TADOConnection 对象的 Execute 操作方法可以运行指定的命令(Command)，其函数声明如下：

```
function Execute(const CommandText: WideString; ExecuteOptions:
          TExecuteOptions = []): _RecordSet; overload;
procedure Execute(const CommandText: WideString; var
          RecordsAffected: Integer; ExecuteOptions:
          TExecuteOptions = [eoExecuteNoRecords]); overload;
```

Open 操作方法

当 TADOConnection 对象的 LoginPrompt 属性值为 True 时，在登录数据库前系统会出现

一个登录对话框，通过登录对话框可以获取用户的帐号与密码。同样的，也可以通过 Open 操作方法在代码中指定用户帐号与密码，但必须先将 LoginPrompt 属性值设为 False。

范例 20-5

目的：本范例的目的是说明如何使用 TADOConnection 对象的 Open 操作方法在代码中指定用户帐号与密码，以连接 Oracle 8i 数据库。

STEP 1 本范例修改自范例 20-2，请先取消步骤 4 所设置的用户帐号与密码。

STEP 2 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I : Integer;
begin
  Try
    //连接数据库
    with ADOConnection1 do
        begin
          Close;
          LoginPrompt := False;
          Open('scott','tiger');
          //打开所有的 DataSet
          For I:= 0 to DataSetCount -1 do
             DataSets[i].Open;
        end;
  except
    //处理异常
    on E : Exception do
      begin
        ShowMessage(E.Message);
        Halt;
      end;
    end;
end;
```

STEP 3 保存工程，按下【F9】功能键重新运行工程。此时系统与范例 20-2 相同会显示数据表的内容。

20.2.3 TADOConnection 对象事件处理

以下就 TADOConnection 对象各种常用的事件加以说明之。

OnWillConnect 事件

当 TADOConnection 对象与数据库服务器进行连接前会触发 OnWillConnect 事件，例如

设置 Active 属性为 True 或是调用 Open 操作方法等。

- *OnConnectComplete* 事件

当 TADOConnection 对象与数据库服务器进行连接完成后会触发 OnConnectComplete 事件。

- *BeforeConnect* 事件

在 TADOConnection 对象与数据库服务器的连接创建之前会触发 BeforeConnect 事件。

- *AfterConnect* 事件

在 TADOConnection 对象与数据库服务器的连接创建之后会触发 AfterConnect 事件。

- *OnDisconnect* 事件与 *AfterDisconnect* 事件

TADOConnection 对象在连接被取消后会触发 OnDisconnect 事件与 AfterDisconnect 事件。

- *BeforeDisconnect* 事件

TADOConnection 对象在连接被取消前会触发 OnBeforeDisconnect 事件。

- *OnWillExecute* 事件

当 TADOConnection 对象的 Execute 操作方法被调用时，在运行命令前会触发 OnWillExecute 事件。

- *OnExecuteComplete* 事件

当 TADOConnection 对象的 Execute 操作方法被调用而且完成命令运行后会触发 OnExecuteComplete 事件。

- *OnBeginTransComplete* 事件

当 TADOConnection 对象的 BeginTrans 操作方法被调用而且完成事务的启动后会触发 OnBeginTransComplete 事件。

- *OnCommitTransComplete* 事件

当 TADOConnection 对象的 CommitTrans 操作方法被调用而且完成事务的确认后会触发 OnCommitTransComplete 事件。

- *OnRollbackTransComplete* 事件

当 TADOConnection 对象的 RollbackTrans 操作方法被调用而且完成事务的取消(恢复)后会触发 OnRollbackTransComplete 事件。

范例 20-6

目的：本范例的目的是说明如何使用 TADOConnection 对象与事务有关的操作方法及事件处理。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 20-2 的 Oracle Server 连接设置。

步骤 3 在 TForm 对象中另外加入一个 TADOTable 对象、一个 TDataSource 对象、TDBGrid 对象、两个 TListBox 对象、两个 TLabel 对象及五个 TButton 对象，其相关属性设置如表 20.5 所示。

表 20.5

对象	属性名称	属性设置值
TADOTable	Name	ADOTable1
	Connection	ADOConnection1
	TableName	EMP
	Active	True
TDataSource	Name	DataSource1
	DataSet	ADOTable1
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TListBox	Name	ListBox1
TListBox	Name	ListBox2
TLabel	Name	Label1
	Caption	运行的命令
TLabel	Name	Label2
	Caption	触发的事件
TButton	Name	btStartTransaction
	Caption	StartTransaction
TButton	Name	btCommit
	Caption	Commit
TButton	Name	btRollback
	Caption	Rollback
TButton	Name	btTableRefresh
	Caption	TableRefresh

步骤 4 在 btStartTransaction 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.btStartTransactionClick(Sender: TObject);
begin
```

```
    //打开一个事务
    ADOConnection1.BeginTrans;
    //显示事务开始
    ListBox1.Items.add('BeginTrans');
  end;
```

5 在 btDelete 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.btDeleteClick(Sender: TObject);
  begin
    //记录删除
    ListBox1.Items.add(ADOTable1.FieldByname('EMPNO').
                                  AsString + ' 员工数据已被删除');
    //删除一条数据
    ADOTable1.delete;
  end;
```

6 在 btCommit 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.btCommitClick(Sender: TObject);
  begin
    //确认所有事务
    ADOConnection1.CommitTrans;
    //显示恢复信息
    ListBox1.Items.add('CommitTrans');
  end;
```

7 在 btRollback 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.btRollbackClick(Sender: TObject);
  begin
    //恢复所有被删除的数据
    ADOConnection1.RollbackTrans;
    //显示恢复信息
    ListBox1.Items.add('RoolbackTrans');
  end;
```

8 在 btTableRefresh 对象的 onClick 事件处理过程中加入以下的代码：

```
  procedure TForm1.btTableRefreshClick(Sender: TObject);
  begin
    //重新显示数据
    ADOTable1.Active := False;
    ADOTable1.Active := True;
  end;
```

9 在 ADOConnection 对象的 onBeginTransComplete 事件处理过程中加入以下的代码：

```
  procedure TForm1.ADOConnection1BeginTransComplete(
       Connection: TADOConnection; TransactionLevel: Integer;
```

```
    Const Error: Error; var EventStatus: TEventStatus);
begin
  //显示事件名称
  ListBox2.Items.add('onBeginTransComplete');
end;
```

Step 10 在 ADOConnection 对象的 onCommitTransComplete 事件处理过程中加入以下的代码：

```
procedure TForm1.ADOConnection1CommitTransComplete(
        Connection: TADOConnection; const Error: Error;
        var EventStatus: TEventStatus);
begin
  //显示事件名称
  ListBox2.Items.add('onCommitTransComplete');
end;
```

Step 11 在 ADOConnection 对象的 onRollbackTransComplete 事件处理过程中加入以下的代码：

```
procedure TForm1.ADOConnection1RollbackTransComplete(
      Connection: TADOConnection; const Error: Error;
       var EventStatus: TEventStatus);
begin
  //显示事件名称
  ListBox2.Items.add('onRollbackTransComplete');
end;
```

Step 12 保存工程。

Step 13 按【F9】功能键运行工程。工程运行后，您可以依次运行 StartTransaction 按钮以打开一个事务、删除几条数据，并决定是否确认事务(Commit)，或是恢复(Rollback)所有修改的数据，最后再单击“Refresh”按钮以重新显示数据，如图 20.19 所示。



图 20.19 TADOConnection 对象的事务处理

20.3 TADODataSet 对象

20.3.1 通过 TADODataSet 对象获取数据

TADODataSet 对象与其他 BDE 数据集组件一样都继承自 TDataSet 对象，因此与 TDataSet 对象有关的属性、操作方法及事件处理请参考第 18 章的详细说明，在此不再赘述。TADODataSet 对象通过 CommandText 属性来获取指定的表名称、存储过程或以 SELECT 为主的 SQL 语句所获取的数据集(DataSet)。TADODataSet 对象只允许我们使用以 SELECT 为主的 SQL 语句，而不支持 DELETE、INSERT 或 UPDATE 等数据操作语言(DML，Data Manipulation Language)，以下就 TADODataSet 对象特有的属性及操作方法加以说明。

- *Connection* 属性

TADODataSet 对象 Connection 属性代表其指定的 TADOConnection 对象。在一个工程中，我们通常会通过相同的 TADOConnection 对象与数据库连接，而非个别使用 ConnectionString 属性来获取数据库连接。

- *ConnectionString* 属性

TADODataSet 对象 ConnectionString 属性允许我们个别为其指定相关的连接信息，详细设置方法请参考 TADOConnection 对象的 ConnectionString 属性。

- *CursorLocation* 属性

TADODataSet 对象的 CursorLocation 属性与 TADOConnection 对象的 CursorLocation 属性同样都是用来决定与数据库连接时所使用的光标(cursor)类型。CursorLocation 属性包含客户端光标(clUseClient)及服务端光标(clUseServer)两种值，详细说明请参考 TADOConnection 对象 CursorLocation 属性的详细说明。

- *CursorType* 属性

TADODataSet 对象 CursorType 属性继承自 TCursorType 类，其包含 ctUnspecified、ctOpenForwardOnly、ctKeyset、ctDynamic 及 ctStatic 等可能值。ctUnspecified 代表 CursorType(光标类型)尚未指定。ctOpenForwardOnly 代表打开一个只能向前(forward-only)的光标，也就是您不能使用 Prior 操作方法将光标往后移，此方法会有比较好的运行效率。ctKeyset 代表打开一个 KeySet 类型的光标，也就是表示其他用户进行的修改并不会实时的更新客户端的数据内容。ctDynamic 代表打开一个 Dynamic 类型的光标，也就是表示其他用户进行的修改会实时更新，而且可以任意操作光标，不管是向前(forward)或往后(back)。ctStatic 代表打开一个 Static 类型的光标，此种光标类型通常使用于报表的制作，其他用户的数据修改并不会进行实时的更新。

- *CommandText* 属性

TADODataSet 对象 CommandText 属性代表所要运行的命令语句，但其必须配合

CommandType 属性而定。例如当 CommandType 属性值为 cmdText，则代表 CommandText 属性值必须为以 SELECT 为主的 SQL 语句。假如 CommandType 属性值为 cmdTable，则代表 CommandText 属性值必须是一个已经存在的表(Table)名称。

CommandType 属性

TADODataSet 对象 CommandType 属性值代表所运行的命令类型，其包含 cmdUnknown、cmdText、cmdTable、cmdStoredProc、cmdFile 及 cmdTableDirect 等可能值。cmdUnknown 代表命令类型尚未指定。cmdText 代表命令类型为 SQL 语句。cmdTable 代表命令类型为指定的数据表。cmdStoredProc 代表命令类型为指定的存储过程。cmdFile 代表命令类型为指定的文件名称。cmdTableDirect 代表命令类型为指定的表名称，而且返回所有的字段数据。

CommandTimeOut 属性

CommandTimeOut 属性用来设置 TADODataSet 对象最长的命令运行等待时间，其默认值为 30 秒。

LockType 属性

CommandTimeOut 属性代表 TADODataSet 对象运行命令所采用的 Lock 类型，其包含 ltUnspecified、ltReadOnly、ltPessimistic、ltOptimistic 及 ltBatchOptimistic 等可能值。ltUnspecified 代表 Lock 类型尚未指定。ltReadOnly 代表此 TADODataSet 对象以只读的方式打开，因此无法运行新增、修改或删除等功能。ltPessimistic 代表当数据源的数据正在更新时，系统会临时锁住其他用户的操作，以确保数据的一致性。ltOptimistic 代表当数据源的数据正在更新时，系统不会锁住其他用户的操作，因此其他用户同样可以对数据进行插入、修改或删除等操作。LtBatchOptimistic 通常使用于批量更新操作中，关于批量更新操作请参考第 20.3.2 节详细说明。

Acitve 属性

TADODataSet 对象的 Active 属性与其他 BDE 数据集组件的 Active 属性一样，都是用来判断或设置此 DataSet 对象是否为打开状态，当 Active 属性值为 True，表示此 DataSet 对象正处于打开状态，您也可以使用 Open 操作方法来打开 TADODataSet 对象。

Parameters 属性

当 TADODataSet 对象 CommandText 属性值中包含参数时，我们必须通过 Parameters 属性来传递参数，其用法请参考 TQuery 对象与 TStoredProc 对象。

GetIndexNames 操作方法

GetIndexNames 操作方法可以获取 TADODataSet 对象所有对应的索引名称，但通常必须在 CommandType 属性值为 cmdTable 才会有效。以下程序片段可以获取 TADODataSet 对象所对应的索引名称(IndexName)。

```
ADODataSet1.GetIndexNames(ListBox1.Items);
```

范例 20-7

目的：本范例的目的是说明如何使用 TADODataSet 对象各种不同获取数据的方法。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 20-3，我们所要连接的数据库为 Microsoft SQL Server。

步骤 3　在 TForm 对象中加入四个 TADODataSet 对象，ADODataSet1 代表使用一般以 SELECT 为主是 SQL 语句。ADODataSet2 代表使用一般以 SELECT 为主是 SQL 语句，但其包含一个输入参数。ADODataSet3 则直接指定数据表(Table)。ADODataSet4 则调用指定的存储过程(Stored Procedure)。请将这四个 TADODataSet 对象的 Connection 属性值都设为 ADOConnection1。

步骤 4　将 ADODataSet1 对象的 CommandType 属性设为 cmdText，而其 CommandText 属性设为“select * from employee”SQL 语句。

步骤 5　将 ADODataSet2 对象的 CommandType 属性设为 cmdText，而其 CommandText 属性设为“select * from employee where job_id = :Job_ID”SQL 语句。

步骤 6　将 ADODataSet3 对象的 CommandType 属性设为 cmdTable，而其 CommandText 属性设为“employee”数据表名称。

步骤 7　将 ADODataSet4 对象的 CommandType 属性设为 cmdStoredProc，而其 CommandText 属性设为“GetEmployee2”存储过程名称。GetEmployee2 是一个存储过程，它有一个代表 Job_ID 的输入参数，而其返回值是符合此 Job_ID 的员工数据。GetEmployee2 存储过程的内容如下所示。

```
CREATE PROCEDURE GetEmployee2 @InJobID int AS
  SELECT Job_Id,Emp_Id,FName,Hire_date
  FROM employee
  WHERE Job_id = @InJobId
```

步骤 8　在 TForm 对象中加入一个 TDataSource 对象与一个 TDBGrid 对象，并将 TDBGrid 对象的 DataSource 属性设为 DataSource1。

步骤 9　另外在 TForm 对象中加入两个 TLabel 对象、一个 TComboBox 对象及一个 TButton 对象，其相关属性设置如表 20.6 所示。

表 20.6

对象	属性名称	属性设置值
Tlabel	Name	Label1
	Caption	选择 TADODataSet 对象
Tlabel	Name	Label2
	Caption	输入参数值
TcomboBox	Name	ComboBox1
	Style	csDropDownList

续上表

对象	属性名称	属性设置值
Tedit	Name	Edit1
	Caption	(空白)

step10 在 TForm 对象的 onCreate 事件处理过程中加入以下的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  //获取 TADOConnection 所关联的 TADODataSet 对象名称
  for I := 0 to (ADOConnection1.DataSetCount-1) do
     Combobox1.Items.Add(ADOConnection1.DataSets[i].Name)
end;
```

step11 在 TComboBox 对象的 onChange 事件处理过程中加入以下代码：

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  //决定是否输入参数
  Edit1.visible := (ADOConnection1.DataSets[
                    ComboBox1.ItemIndex] as TADODataSet).
                    Parameters.count > 0;
end;
```

step12 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
begin
  //先关闭 DataSet 对象
  ADOConnection1.DataSets[ComboBox1.ItemIndex].Active :=

False;
  //设置 DataSource 对象的 DataSet 属性
  DataSource1.DataSet := ADOConnection1.DataSets[
                                    ComboBox1.ItemIndex];
  //设置参数值
  if Edit1.visible then
    begin
      //获取参数的个数
      I := (ADOConnection1.DataSets[ComboBox1.ItemIndex] as
            TADODataSet).Parameters.Count - 1;
      //设置最后一个参数值(因为 TADODataSet 有两个参数,
```

```
    //我们只需设置最后一个即可)
    (ADOConnection1.DataSets[ComboBox1.ItemIndex] as
      TADODataSet).Parameters[0].value := StrToInt(Edit1.text);
  end;

  //启动指定的 TADODataSet 对象
  ADOConnection1.DataSets[ComboBox1.ItemIndex].Active := True;
end;
```

(13) 保存工程。

(14) 按【F9】功能键运行工程，工程运行后，您可以从 ComboBox1 中任意选择所要运行的 TADODataSet 对象，并输入必要的参数，再单击“获取数据”按钮，那么指定的数据就会显示在 TDBGrid 对象中，如图 20.20 所示。



图 20.20　TADODataSet 对象运行范例

DeleteRecords 操作方法

DeleteRecords 操作方法提供您删除 TADODataSet 对象中的记录，此操作方法提供 arCurrent、arFiltered、arAll 及 arAllChapters 等不同的参数，以删除一条或一条以上的记录。arCurrent 代表删除目前光标所在的记录。arFiltered 代表删除符合 Filter 过滤条件的记录。arAll 代表删除目前 TADODataSet 对象中所有的记录。arAllChapters 代表删除 ADO chapters 中所有的记录。

SaveToFile 操作方法

SaveToFile 操作方法允许我们将 TADODataSet 对象所获取的数据以文件保存，目前保存数据的格式有 ADTG 与 XML 两种。

LoadFromFile 操作方法

LoacFromFile 操作方法允许我们将先前以 SaveToFile 操作方法所保存的数据加载 TADODataSet 对象中。

范例 20-8

目的：本范例的目的是说明如何使用 TADODataSet 对象与文件相关的操作方法。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 20-2 的 Oracle Server 连接示范。

步骤 3 在 TForm 对象中加入一个 TADODataSet 对象、一个 DataSource 对象、一个 TDBGrid 对象、一个 TSaveDialog 对象、一个 TOpenDialog 对象及两个 TButton 对象，其相关属性设置如表 20.7 所示。

表 20.7

对象	属性名称	属性设置值
TADODataSet	Name	ADODataSet1
	Connection	ADOConnection1
	CommandText	employee
	CommandType	cmdTable
	Acitve	True
TDataSource	Name	DataSource1
	DataSet	ADODataSet1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TSaveDialog	Name	SaveDialog1
TOpenDialog	Name	OpenDialog1
TButton	Name	Button1
	Caption	Save
TButton	Name	Button2
	Caption	Load

步骤 4 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //将数据写回文件中
  if SaveDialog1.execute then
    ADODataSet1.SaveToFile(SaveDialog1.FileName,pfXML);
end;
```

步骤 5 在 Button2 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //从文件中读数据
```

```
    if OpenDialog1.execute then
      ADODataSet1.LoadFromFile(OpenDialog1.FileName);
  end;
```

6 保存工程。

7 按【F9】功能键运行工程。工程运行后，您可以通过 Save 按钮将数据以 XML 格式保存。也可以通过 Load 按钮从文件中读取先前以 XML 格式保存的文件，如图 20.12 所示。

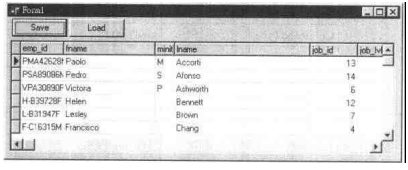


图 20.21　TADODataSet 对象文件访问

20.3.2　*TADODataSet* 对象的批量更新处理

TADODataSet 对象所提供的批量更新处理与 TADOConnection 对象的事务控制在功能上是相同的，您也可以将此批量更新功能视为 TADODataSet 对象层次的事务控制。所谓的批量更新处理是指当 TADODataSet 对象保存变动数据时，要不是全部写回，就是全部取消。一个 TADODataSet 对象要以批量模式打开，必须符合下列条件。

1. TADODataSet 对象的 CursorType 属性值必须是 ctKeySet 或 ctStatic。
2. TADODataSet 对象的 LockType 属性值必须是 ltBatchOptimistic。
3. TADODataSet 对象的 CommandText 属性值必须是以 SELECT 为主的 SQL 语句。相对的，CommandType 属性值必定为 cmdText。

当 TADODataSet 对象以批量模式打开后，系统会保留所有对此 TADODataSet 对象的修改，不会直接对数据库的内容进行更新。您可以使用 UpdateBatch 操作方法来更新修改内容，也可以使用 CancelBatch 操作方法来取消数据的修改。您可以查看 RecordStatus 属性来判断每一条记录的修改状态，以下将以范例来说明如何使用 TADODataSet 对象的批量更新功能。

范例 20-9

目的：本范例的目的是说明如何使用 TADODataSet 对象的批量更新功能。

1 创建新的工程。

2 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 18-2 的 Oracle Server 连接示范。

STEP 2 在 TForm 对象中加入一个 TADODataSet 对象、一个 DataSource 对象、一个 TDBGrid 对象、一个 TDBNavigator 对象、一个 TListBox 对象及两个 TButton 对象，其相关属性设置如表 20.8 所示。

表 20.8

对象	属性名称	属性设置值
TADODataSet	Name	ADODataSet1
	Connection	ADOConnection1
	CommandText	select * from employee
	CommandType	cmdText
	CursorType	ctStatic
	LockType	ltBatchOptimistic
	Acitve	True
TDataSource	Name	DataSource1
	DataSet	ADODataSet1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TDBNavigator	Name	DBNavigator1
	DataSource	DataSource1
TListBox	Name	ListBox1
TButton	Name	Button1
	Caption	批量更新
TButton	Name	Button2
	Caption	取消更新

STEP 3 在 ADODataSet1 的 onBeforeDelete 事件处理过程中加入以下代码：

```
procedure TForm1.ADODataSet1BeforeDelete(DataSet: DataSet);
begin
  //显示信息
  ListBox1.Items.add('Delete Record:' +
DataSet.FieldByname('emp_id').Asstring);
end;
```

STEP 4 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //运行批量更新
  ADODataSet1.UpdateBatch(arAll);
  //显示信息
  ListBox1.Items.add('批量更新');
end;
```

5) 在 Button2 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  //运行批量更新
  ADODataSet1.CancelBatch(arAll);
  //显示信息
  ListBox1.items.add('取消更新');
end;
```

6) 保存工程。

7) 按【F9】功能键运行工程，工程运行后，您可以任意修改任何记录(这里以删除作示范)，当单击“取消更新”按钮时系统会自动取消您所作的修改，您可以通过 DBNavigator1 所提供的“”按钮获取取消更新后的数据，如图 20.22 所示。



图 20.22　TADODataSet 对象的批量更新功能

20.4　TADOTable 对象

TADOTable 对象可以说是 TADODataSet 对象的一个特殊化组件，其使用方法与 DBE 的 TTable 对象相当类似。有关 TTable 对象的详细介绍请参考第 18 章，以下仅以一个简单的范例来说明如何通过 TADOTable 对象来获取数据。

范例 20-10

目的：本范例目的是说明如何使用 TADOTable 对象的数据过滤功能。

1) 创建新的工程。

2) 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 18-2。

3) 在 TForm 对象中加入一个 TADOTable 对象、一个 DataSource 对象、一个 TDBGrid 对象、一个 TDBNavigator 对象及一个 TButton 对象，其相关属性设置如表 20.9 所示。

表 20.9

对象	属性名称	属性设置值
TADOTable	Name	ADOTable1
	Connection	ADOConnection1
	TableName	employee
	Acitve	True
TDataSource	Name	DataSource1
	DataSet	ADOTable1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TButton	Name	Button1
	Caption	数据过滤

步骤 4 在 TButton 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str:String;
  Flag:Boolean;
begin
  //先通过 InputQuery 对象来输入过滤条件
  Str:='';
  Flag:=InputQuery('请入过滤条件','过滤条件',Str);
  if Flag then
  begin
    //进行过滤
    Try
      ADOTable1.Filtered:=False;
      ADOTable1.Filter:=Str;
      ADOTable1.Filtered:=True;
    //过滤条件错误
    except
      ShowMessage('过滤条件错误，无法运行');
      ADOTable1.Filtered:=False;
    end;
  end;
end;
```

步骤 5 保存工程。

步骤 6 按【F9】功能键运行工程。工程运行后，您可以单击“数据过滤”按钮，并输入过滤条件，此时 TDBGrid 便会实时显示过滤结果。图 20.23 示范获取 Job_Id 等于 5 的员工数据。

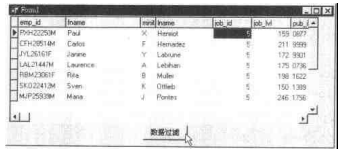


图 20.23　TADOTable 对象的过滤功能

20.5　TADOQuery 对象

TADOQuery 对象可以说是 TADODataSet 对象的一个特殊化组件，其使用方法与 DBE 的 TQuery 对象相当类似。有关 TQuery 对象的详细介绍请参考第 18 章，以下仅以一个简单的范例来说明如何通过 TADOQuery 对象来获取数据。

范例 20-11

目的：本范例目的是说明如何使用 TADOQuery 对象运行 SQL 语句。

1. 创建新的工程。
2. 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 18-2 的 Oracle Server 连接示范。
3. 在 TForm 对象中加入一个 TADOQuery 对象、一个 DataSource 对象、一个 TDBGrid 对象、一个 TMemo 对象、一个 TButton 对象及一个 TCheckBox 对象，其相关属性设置如表 20.10 所示。

表 20.10

对象	属性名称	属性设置值
TADOQuery	Name	ADOQuery1
	Connection	ADOConnection1
TDataSource	Name	DataSource1
	DataSet	ADOQuery1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TMemo	Name	Memo1
TButton	Name	Button1
	Caption	运行 SQL 语句
TCheckBox	Name	CheckBox1
	Caption	会返回数据集

STEP 4 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //先关闭 ADOQuery 对象
  ADOQuery1.Close;

  //指定 SQL 语句
  ADOQuery1.SQL.Text := Memo1.Text;

  if CheckBox1.Checked then
    //SQL 语句会返回数据集
    ADOQuery1.Open
  else
    //SQL 语句不会返回数据集
    ADOQuery1.ExecSQL;
end;
```

STEP 5 保存工程。

STEP 6 按【F9】功能键运行工程。工程运行后，您可以输入任意的 SQL 语句，包括 SELECT、INSERT、DELETE、UPDATE 及 CREATE 等语句。如果您的 SQL 语句会返回数据集，那么我们会使用 Open 操作方法来打开 TADOQuery 对象。反之，如果 SQL 语句不会返回数据集，则使用 ExecSQL 操作方法来打开 TADOQuery 对象。图 20.24 示范如何通过 SQL 语句从数据库中获取指定的数据内容。

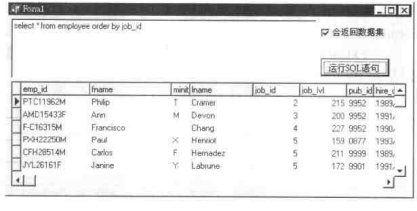


图 20.24 使用 TADOQuery 对象运行 SQL 语句

20.6 TADOStoredProc 对象

TADOStoredProc 对象可以说是 TADODataSet 对象的一个特殊化组件，其使用方法与 DBE 的 TStoredProc 对象相当类似。有关 TStoredProc 对象的详细介绍请参考第 18 章，以下仅以一个简单的范例来说明如何通过 TADOStoredProc 对象来获取数据。

范例 20-12

目的：本范例目的是说明如何使用 TADOStoredProc 对象来运行存储过程。本范例要调用的 GetEmployee2 存储过程是我们在范例 20-7 中所创建的，它有一个代表 Job_ID 的输入参数，而其返回值是符合此 Job_ID 的员工数据，详细说明请参考范例 20-7。

step 1 创建新的工程。

step 2 在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 20-3，我们所要连接的数据库为 Microsoft SQL Server。

step 3 在 TForm 对象中加入一个 TADOStoredProc 对象、一个 DataSource 对象、一个 TDBGrid 对象、一个 TLabel 对象、一个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 20.11 所示。

表 20.11

对象	属性名称	属性设置值
TADOStoredProc	Name	ADOStoredProc1
	Connection	ADOConnection1
	ProcedureName	GetEmployee2
TDataSource	Name	DataSource1
	DataSet	ADOStoredProc1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TLabel	Name	Label1
	Caption	输入参数值
TEdit	Name	Edit
	Text	(空白)
TButton	Name	Button1
	Caption	获取数据

step 4 在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //先关闭 ADOStoredProc 对象
  ADOStoredProc1.Close;
  '设置参数
  ADOStoredProc1.Parameters.ParamByName('@InJobID').
                          Value := StrToInt(Edit1.Text);
  //打开 TADOStoredproc 对象
  ADOStoredProc1.Open;
end;
```

步骤 5 保存工程。

步骤 6 按【F9】功能键运行工程。工程运行后，您可以输入指定的 Job_Id，再单击“获取数据”按钮以运行指定的存储过程，其结果如图 20.25 所示。

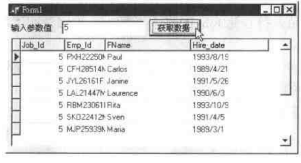


图 20.25 使用 TADOStoredProc 对象运行存储过程

20.7 TADOCommand 对象

TADOQuery 对象可以运行任何的数据处理语言(DML、Data Manipulation Language)，无论该 SQL 语句是否会返回数据集，TADODataSet 对象其实也可以运行类似于 TADOQuery 对象的功能。为什么 ADO 组件要提供这么多相类似的组件呢？事实上每一个组件在运行同一种命令时会使用不同的方法，因此其运行效率也会有些稍微的不同。您可以根据各种 ADO 组件的不同特性来选择运行适当的命令。本小节将为大家介绍 ADO 组件中另一个重要的组件，那就是 TADOCommand 对象。TADOCommand 对象提供运行命令的能力，一次一个命令，这里所谓的命令包含数据定义语言(DDL, Data Definition Language)与数据处理语言。

TADOCommand 对象虽然被设置来运行不会返回数据集的 SQL 语句，但它也允许您使用 Execute 操作方法来运行有返回数据集的 SQL 语句。以下就 TADOCommand 对象特有且重要的属性及操作方法加以说明。

CommandText 属性

TADOCommand 对象 CommandText 属性代表所要运行的命令语句，但其必须配合 CommandType 属性而定。例如当 CommandType 属性值为 cmdText，则代表 CommandText 属性值必须为以 SELECT 为主的 SQL 语句。假如 CommandType 属性值为 cmdTable，则代表 CommandText 属性值必须是一个已经存在的表(Table)名称。

CommandType 属性

TADOCommand 对象 CommandType 属性值代表所运行的命令类型，其包含 cmdUnknown、cmdText、cmdTable、cmdStoredProc、cmdFile 及 cmdTableDirect 等可能值。cmdUnknown 代表命令类型尚未指定。cmdText 代表命令类型为 SQL 语句。cmdTable 代表命令类型为指定的数据表。cmdStoredProc 代表命令类型为指定的存储过程。cmdFile 代表命令类型为指定的文件名称。cmdTableDirect 代表命令类型为指定的表名称，而且返回所有的字

段数据。

Execute 操作方法

您可以通过调用 Execute 操作方法来运行 TADOCommand 对象的命令。Execute 操作方法有“数据集返回值”与“无数据集返回值”两种，如果 TADOCommand 对象所要运行的命令是“无数据集返回值”那么您可以直接调用 Execute 操作方法。反之，如果该命令是有“数据集返回值”，那么 Execute 操作方法会有一个 RecordSet 的返回值，因此您可以使用下列语法来获取返回值。

```
ADODataSet1.RecordSet := ADOCommand1.Execute;
```

Cancel 操作方法

Cancel 操作方法可以中断 Execute 操作方法的运行。

范例 20-13

目的：本范例的目的是说明如何使用 TADOCommand 对象来运行 SQL 语句。

步骤 1　创建新的工程。

步骤 2　在 TForm 对象中加入一个 TADOConnection 对象，其相关属性设置请参考范例 20-2 的 Oracle Server 连接示范。

步骤 3　在 TForm 对象中加入一个 TADOCommand 对象、一个 TADODataSet 对象、一个 DataSource 对象、一个 TDBGrid 对象、一个 TMemo 对象、一个 TButton 对象及一个 TCheckBox 对象，相关属性设置如表 20.12 所示。

表 20.12

对象	属性名称	属性设置值
TADOCommand	Name	ADOCommand1
	Connection	ADOConnection1
TADODataSet	Name	ADODataSet1
TDataSource	Name	DataSource1
	DataSet	ADODataSet1
TDBGrid	Name	DBGrid
	DataSource	DataSource1
TMemo	Name	Memo1
TButton	Name	Button1
	Caption	运行 SQL 语句
TCheckBox	Name	CheckBox1
	Caption	会返回数据集

步骤 4　在 Button1 的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //指定 SQL 语句
  ADOCommand1.CommandText := Memo1.Text;
  if CheckBox1.Checked then
    //SQL 语句会返回数据集
    ADODataSet1.RecordSet := ADOCommand1.Execute
  else
    //SQL 语句不会返回数据集
    ADOCommand1.Execute;
end;
```

5 保存工程。

6 按【F9】功能键运行工程。工程运行后，您可以输入任意的 SQL 语句，包括数据定义语言与数据处理语言。图 20.26 示范如何通过 SQL 语句从数据库中获取指定的数据内容。

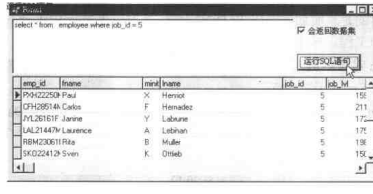


图 20.26 使用 TADOCommand 对象运行 SQL 语句

第 21 章

Delphi 6 多线程程序

本章目标

本章将为您介绍 Delphi 6 的多线程(Multi-Thread)程序。线程(Thread)又称执行控制绪，您可以想象线程就好像是一个指针，它会指着正在运行的那一行程序。程序一行一行的往下运行，而指针也会一行一行的往下移，直到程序结束为止。因此，线程负责控制整个程序的运作与运行。而多线程(Multi-Thread)则是指在同一个程序中、同一个时间点、同时有好几个线程运作着。每个线程彼此独立，而且互不干扰地运行着。

本章主题

21.1 多线程(Multi-Thread)简介

21.2 第一个多线程程序

21.3 多线程与文件复制

21.4 多线程与数据库应用程序

21.1 多线程(Multi-Thread)程序简介

在讨论 Delphi 6 的多线程(Multi-Thread)程序之前，我们先来看看一个传统循环的程序范例。

范例 21-1

目标：本范例将示范一个传统循环的程序范例。

步骤 1 创建新的工程。

步骤 2 在 TForm 对象中加入两个 TGroupBox 对象，其默认名称分别为 GroupBox1 与 GroupBox2。这两个 TGroupBox 对象分别代表 TForm 对象上的两个工作项目，如图 21.1 所示。

步骤 3 为 GroupBox1 中 TButton 对象的 onClick 事件处理过程中加入以下的代码。这段代码主要是用来累计两个 10000 次循环，并计算其运算的时间再加以显示。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i, j, TotalMilliSecond : Integer;
  sum : Extended;   //循环累计变量
  StartTime : TDateTime; //记录开始的时间
begin
  //获取开始计算的时间
  StartTime := Now;
  //累计循环的计算值
  sum := 0;
  For i := 1 to 10000 do
    For j := 1 to 10000 do
        sum := sum + (i + j);
  //显示累计结果
  Edit1.Text := FloatToStr(Sum);
  //获取计算的时间
  TotalMilliSecond := MilliSecondsBetween(StartTime,Now);
  //显示计算的时间
  LabelTotalMilliSecond.Caption := IntToStr(TotalMilliSecond);
end;
```

步骤 4 保存并按【F9】键运行工程。工程运行后，当您单击 TButton 按钮时，系统会将程序的控制权交给步骤 3 的循环。在循环尚未结束之前您无法在这个程序上做其他任何的事，如图 21.1 所示。



图 21.1　非线程程序范例

注意

虽然笔者在前面的章节中曾经说过 TApplication 对象的 ProcessMessage 操作方法，可以让程序在运行中接收其他的系统信息。但通过笔者的测试，此方法会严重影响系统的总运行时间。因此，笔者不认为它可以完全解决上述问题。

因为上面的范例是一个单线程(Single-Thread)应用程序，因此，在线程运行完毕之前不会交出程序的控制权。这就是在循环尚未结束之前，您无法在这个程序中做其他事情的主要原因。为了解决这个问题因此便衍生出多线程(Multi-Thread)的应用程序，一般来说，一个多线程的应用程序包含下列几个特点：

- 充分利用 CPU 性能，避免产生瓶颈。在一般的情况下，CPU 的处理速度要远比 I/O 处理速度快很多。假设某个应用程序的 I/O 与 CPU 的处理量各占 50%，这对一个单线程的应用程序而言，CPU 会有很多的闲置时间，因为它必须等待 I/O。这些 I/O 的操作包括访问磁盘上的文件、访问网络资源或显示多媒体数据等。在整个工作完成前 CPU 一直处于等待状态，而 I/O 则处于非常忙碌的状态。多线程的程序可以采用分时多任务有效的解决上述问题，并且充分使用 CPU 性能以避免产生瓶颈。
- 按优先级处理指定工作。优先级(Pripority)是指应用程序获取使用 CPU 的权限高低，一个高优先级的线程可以用来处理对时间很关键的应用程序。相对的，其他的批处理操作就可以使用较低优先级的线程来完成它。一般的应用程序无法控制优先级的问题，但多线程程序可以在线程被创建的同时，指定该线程的优先级。
- 利用多处理器(CPU)以改善运行效率。在单 CPU 的计算机上运行多线程，只能算是分时多任务。也就是说，所有的线程必须依照优先级高低的不同，分配不同的 CPU 使用时间。反之，如果您的计算机拥有多个 CPU，而且也拥有支持多 CPU 的操作系统，例如:Windows NT，那么多线程可以运行的更有效率。

注意

多任务处理(Multi-Process)并不等于多线程(Multi-Thread)。因为多任务处理属于操作系统层面，它可以让同一台计算机在同一个时间点，运行多个程序。当然，如果您的计算机只有一个 CPU 的话，它只能作到所谓的分时(Time-Sharing)多任务，所谓分时多任务，是指操作系统以很短的时间不停地转移程序的运行权，让用户觉得好像是很多程序同时被运行一样。多线程(Multi-Thread)与多任务处理(Multi-Process)一样，也是为了让计算机能同时处理很多件事情，但最大不同在于前者的所有线程都来自同一个程序。所以，您可以说，“多任务处理是指多个程序一起运行，多线程则是将程序分成好几个部分一起运行”。

相信通过上面的说明后，您应该已经对 Delphi 的多线程应用程序有了更进一步的了解。以下笔者将说明如何编写 Delphi 的多线程。

定义多线程对象

Delphi 6 将多线程对象封装成 TThread 对象，您只要通过 IDE 的几个简单步骤就可轻易定义多线程对象。下列程序片段示范笔者所自行定义的多线程对象，其名称为 TMyFirstThread。

```
type
  TMyFirstThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;
```

设置线程的优先级

Delphi 6 所提供的多线程对象提供七种不同的优先级。您可以在调用线程时，同时指定其优先级，也可以通过改写多线程对象的构造函数来设置该多线程的优先级。

```
type
  TMyFirstThread = class(TThread)
(略)
  protected
(略)
    constructor Create;
  end;
(略)
constructor TMyFirstThread.Create;
begin
  //构造函数的参数
  //True 代表这个线程是可以被暂停的
  //Flase 代表这个线程是无法被暂停的
  inherited Create(True);
  Priority := tpTimeCritical; //设为最高权限
end;
```

Delphi 6 所提供的各种优先级，如表 21.1 所示。

表 21.1

优先级参数	说明
tpIdle	这种线程只会在系统闲置时才会被运行，也就是说，操作系统并不会为了运行优先级为 tpIdle 的线程，而终止其他的线程
tpLowest	这种线程比一般的线程(tpNormal)少了两个等级

续上表

优先级参数	说明
tpLower	这种线程比一般的线程(tpNormal)少了一个等级
tpNormal	这是一般正常的线程
tpHigher	这种线程比一般的线程(tpNormal)多了一个等级
tpHighest	这种线程比一般的线程(tpNormal)多了两个等级
tpTimeCritical	这里线程具有最高权限



操作系统会优先处理具有 tpTimeCritical 优先级的线程，但是它可能引起其他线程处于“等待”的状态。所谓“等待”是指线程因为优先级较低，因而无法获取 CPU 的处理权，所以处于长期等待。

编写线程的运行内容

您可以将线程的运行内容写于 TThread 对象所提供的 Execute 操作方法中。

```
procedure TMyFirstThread.Execute;
begin
  //将线程的内容写在这里
end;
```

指定线程的结束函数

您可以在线程对象被创建时，同时指定线程对象 onTerminate 的事件处理过程函数。

```
procedure TForm1.ButtonRUNClick(Sender: TObject);
var
  Thread1 : TMyFirstThread;
begin
  Thread1 := TMyFirstThread.Create;
  Thread1.OnTerminate:=Thread1Done;
end;
```

结束线程对象

TThread 对象的 Terminate 操作方法可结束指定的线程对象、Suspend 操作则可暂时停止指定的线程对象、Suspended 属性可用来判断指定的线程是否处于暂停状态、Resume 操作方法可继续一个被暂停的线程对象。

21.2　第一个多线程程序

在这个小节中，笔者将为您示范如何编写第一个多线程程序。



21-2

目标：本范例将示范如何以多线程的方式改写范例 21-1。

1. 创建新的工程。
2. 在 TForm 对象中加入两个 TGroupBox 对象，其默认名称分别为 GroupBox1 与 GroupBox2。这两个 TGroupBox 对象分别代表 TForm 对象上的两个工作项目，如图 21.2 所示。
3. 从“File/New/Other...”下拉式菜单中打开“New Items”对话框，并从“New”页标签中点选“Thread Object”。
4. 请在“New Thread Object”对话框中输入指定的线程名称，并单击“OK”按钮，这里以 TMyFirstThread 为例。



图 21.2 输入指定的线程名称

5. 此时系统自动产生一个程序单元文件，并在该单元文件中声明一个名称为 TMyFirstThread 的线程对象，如下所示。

```
unit Unit2;
interface
uses
  Classes;
type
  TMyFirstThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;
implementation

procedure TMyFirstThread.Execute;
begin
  { Place thread code here }
end;
end.
```

注意 如果您的多线程对象内容很简单，可以通过手动的方式将多线程对象与其他的 TForm 对象声明于同一个程序单元文件中，以简化不同程序单元文件间的引用操作。但如果您的多线程对象内容很复杂，笔者还是建议您使用个别的程序单元文件。

6. 在 TMyFirstThread 对象中声明一个名称为 Sum 的私有变量及一个名称为

UpdateCaption 的私有过程。前者是用来保存循环的累计值，而后者则是负责线程的更新操作。

```
type
  TMyFirstThread = class(TThread)
  private
    { Private declarations }
    Sum : Extended;  //保存循环的累计值
    procedure UpdateCaption; //负责线程的更新操作
```

步骤 7　UpdateCaption 过程是笔者用来更新线程的运算结果，因此您可以自行更改此过程的名称。请为此过程填入下列的代码，这一段代码主要是将线程的累计结果显示于 TForm1 上的 Edit1 对象中，因此您必须引用 TForm1 所属的程序单元文件。另外，因为我们使用 FloatToStr 函数，所以您还必须引用 SysUtils 程序单元文件。

```
procedure TMyFirstThread.UpdateCaption;
begin
   //use  Unit1 and SysUtils
   Form1.Edit1.Text := FloatToStr(Sum);;
end;
```

步骤 8　为 Execute 程序加入以下的代码，这一般代码主要是累计两个 10000 次的循环，并且通过 synchronize 操作方法来更新线程的运算结果。

```
procedure TMyFirstThread.Execute;
var
  i, j : Integer;
begin
  //累计循环的计算值
  sum := 0;
  For i := 1 to 10000 do begin
    For j := 1 to 10000 do begin
       sum := sum + (i + j);
    end;
    if Terminated then Exit;  //假如收到终止命令则结束
    synchronize(UpdateCaption); //更新线程的运算结果
  end;
end;
```



Synchronize 操作方法可以用来更新线程的运算结果，其输入参数为指定的过程名称。例如 UpdateCaption 就是我们在步骤 7 所创建的过程。Synchronize 操作方法与 Application.Process Message 操作方法一样会严重影响到线程的总运算时间，因此 Synchronize 操作方法的调用位置要特别的留意。

步骤 9　到目前为止，我们已经完成多线程对象的相关声明工作。接下来笔者要示范如何调用 TMyFirstThread 线程对象。请为 TForm 对象声明下列变量与过程。

```
TForm1 = class(TForm)
```

```
(略)
 public
 StartTime : TDateTime; //记录线程开始运行的时间
 Thread1 : TMyFirstThread;//声明一个TmyFirstThread类型的变量
 //声明一个过程，当线程结束时会调用此函数
 procedure Thread1Done(Sender : TObject);
 end;
```

(10) 为 Thread1Done 程序加入下列的代码，这段代码主要是计算线程所消耗的时间并加以显示。

```
procedure TForm1.Thread1Done(Sender : TObject);
var
  TotalMilliSecond : Integer;
begin
  //获取计算的时间
  TotalMilliSecond := MilliSecondsBetween(StartTime,Now);
  //显示计算的时间
  Label1.Caption := IntToStr(TotalMilliSecond);
end;
```

(11) 在 TButton1 对象的 onClick 事件处理过程中加入以下的代码，这段代码主要的目标是动态创建一个线程对象，并设置其相关的属性及事件处理过程。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //获取开始计算的时间
  StartTime := Now;
  //动态创建一个线程对象
  Thread1 := TMyFirstThread.Create(False);
  //设置运行权限
  Thread1.Priority := tpNormal;
  //设置线程的 onTerminate 事件处理过程
  Thread1.OnTerminate:=Thread1Done;
  //当线程结束时自动释放
  Thread1.FreeOnTerminate := True;
end;
```

(12) 按【F9】运行工程。工程运行后，当您按下 Button1 按钮时，程序会动态创建线程并随时显示其运算结果。最重要的是，您可以同时做其他的工作，例如输入的工作。如图 21.3 所示。



（线程运算中）



（线程运算结束）

图 21.3　多线程范例



您可能会觉得很奇怪，为什么范例 21-1 与范例 21-2 两个范例所用的毫秒数竟然相差这么多呢?除了，线程对象本身的创建与释放时间外，其最主要的时间来自 synchronize 操作方法。就像笔者前面所说，synchronize 操作方法非常占用线程的总运行时间，因此您不可轻视 synchronize 操作方法的调用位置。

经过范例 21-2 的说明，相信您应该已经了解如何编写简单的多线程程序。在接下来的范例中，笔者将为您示范如何在程序运行中暂停或终止指定的线程对象。



21-3

目标：本范例将修改自范例 21-2，示范如何暂停或终止指定的线程对象。

(1) 打开范例 21-2。

(2) 在 TForm 对象中加入两个 TButton 对象，其相关属性设置如表 21.2 所示。

表 21.2

对象	属性名称	属性设置值
TButton	Name	ButtonSTOP
	Caption	停止计算
TButton	Name	ButtonSuspend
	Caption	暂停计算

(3) 在 ButtonSuspend 对象的 onClick 事件处理过程加入下列的代码，这段程序主要是暂停或继续指定的线程对象。

```
if ButtonSuspend.Caption = '暂停计算' then begin
  Thread1.Suspend;
  ButtonSuspend.Caption := '继续计算';
end else begin
  Thread1.Resume;
  ButtonSuspend.Caption := '暂停计算';
end;
```

(4) 在 ButtonSTOP 对象的 onClick 事件处理过程加入下列的代码，这段程序主要是终止

指定的线程。假如您指定的线程处于暂停状态，是无法直接结束此线程的。您必须先继续线程的运行，然后再终止它。

```
procedure TForm1.ButtonSTOPClick(Sender: TObject);
begin
  //假将处于暂停状态，是无法直接结束此线程的
  if Thread1.Suspended then ButtonSuspendClick(nil);
  //结束线程
  Thread1.Terminate;
end;
```

5 保存并按【F9】运行工程。工程运行后，当线程被启动后，您可任意的暂停或终止线程的运行，如图 21.4 所示。



图 21.4 终止多线程的运行

前面几个范例中，笔者示范如何编写多线程程序，也示范了如何暂停与终止指定的线程。接下来的范例中，笔者要示范如何在一个工程中共享相同的线程对象，并将其运行结果显示于不同类别的对象，例如 TEdit 对象、TProgressBar 对象与 TGauge 对象。

范例 21-4

目标：示范如何在一个工程中共享相同的线程对象，并将其运行结果显示于不同类别的对象。

1 创建新的工程。

2 在 TForm 对象中加入三个 TGroupBox 对象，其默认名称分别为 GroupBox1、GroupBox2 与 GroupBox3。这三个 TGroupBox 对象分别代表 TForm 对象上的三个工作项目，如图 21.5 所示。

3 分别为三个 TGroupBox 对象加入 TLable 对象、TEdit 对象、TProgressBar 对象、TGauge 对象及 TButton 对象，其相关属性设置如表 21.3 所示。

表 21.3

对象	属性名称	属性设置值
TLabel	Name	Label1
	Caption	计算结果

续上表

对象	属性名称	属性设置值
TLabel	Name	Label2
	Caption	计算结果
TLabel	Name	Label3
	Caption	计算结果
TButton	Name	ButtonRUN1
	Caption	运行计算
TButton	Name	ButtonRUN2
	Caption	运行计算
TButton	Name	ButtonRUN3
	Caption	运行计算
TEdit	Name	Edit1
TProgressBar	Name	ProgressBar1
	Max	100000
TGauge	Name	Gauge1
	ForeColor	clBlue

STEP 4 从“File/New/Other...”下拉式菜单中打开“New Items”对话框，并从“New”页标签中点选“Thread Object”。

STEP 5 请在“New Thread Object”对话框中输入指定的线程名称，并单击“OK”按钮，这里以 TMyFirstThread 为例。此时系统自动产生一个程序单元文件，并在该单元文件中声明一个名称为“TMyFirstThread”的线程对象。

STEP 6 在 TMyFirstThread 对象中声明两个私有的整数变量，其名称为 Flag 及 Positon。Flag 是用来判断目前的线程对象是通过哪一种构造函数所创建的，Position 是用来记录目前循环的位置，也就是指循环的进度。另外再加入 TEdit、TProgressBar 及 TGauge 等三种对象变量，这些对象变量将用于显示线程的运算结果或进度。最后再声明一个名称为 UpdateCaption 的过程及二个不同参数的构造函数，其中 UpdateCaption 过程负责线程的更新操作。

```
type
  TMyFirstThread = class(TThread)
  private
    Flag : Integer;      //判断使用哪一种构造函数
    Position : Integer; //记录目前循环的位置
    myEdit:TEdit;        //TEdit 对象(配合构造函数 1 使用)
    //TProgressBar 对象(配合构造函数 3 使用)
    myProgressBar : TProgressBar;
    myGauge:TGauge;      //TGauge 对象(配合构造函数 2 使用)
    procedure UpdateCaption;  //更新函数
```

```
  protected
    procedure Execute; override;  //运行线程函数
  public
    //第一种构造函数，其参数为 TEdit 对象
    Constructor Create(varEdit:TEdit); overload;
    //第二种构造函数，其参数为 TProgressBar 对象
    Constructor Create(varProgressBar:TProgressBar); overload;
    //第三种构造函数，其参数为 TGauge 对象
    Constructor Create(varGauge:TGauge); overload;
  end;
```

步骤 7 为上一步骤所声明的三种构造函数与 UpdateCaption 过程分别加入指定的代码：

```
//第一种构造函数，其传入对象为 TEdit 对象
Constructor TMyFirstThread.Create(varEdit:TEdit);
begin
    inherited Create(False);
    MyEdit:=varEdit;  //指定显示进度的对象
    FreeOnTerminate:=True;
    Flag := 1; //通过 Flag 变量来识别线程对象
end;

//第二种构造函数，其传入对象为 TProgressBar 对象
Constructor TMyFirstThread.Create(varProgressBar:TProgressBar);
begin
    inherited Create(False);
    myProgressBar := varProgressBar; //指定显示进度的对象
    FreeOnTerminate:=True;
    Flag := 2;  //通过 Flag 变量来识别线程对象
end;

//第三种构造函数，其传入对象为 TGauge 对象
Constructor TMyFirstThread.Create(varGauge:TGauge);
begin
    inherited Create(False);
    myGauge := varGauge;  //指定显示进度的对象
    FreeOnTerminate:=True;
    Flag := 3; //通过 Flag 变量来识别线程对象
end;

//运行同步更新的自定义函数
procedure TMyFirstThread.UpdateCaption;
begin
   //根据不同的传入参数显示运行结果
   case Flag of
     1 : Form1.Edit1.Text := MyEdit.Text;
```

```
      2 : Form1.ProgressBar1.Position := Position;
      3 : Form1.Gauge1.Progress := (Position * 100 ) div 100000 ;
    end;
end;
```

步骤 8 为线程的 Execute 过程加入以下的代码。无论您是调用哪一种构造函数所产生的线程对象都会使用这个 Execute 过程。

```
procedure TMyFirstThread.Execute;
var
  i : Integer;     //循环变量
  sum : Extended; //加总变量为 Local 变量
begin
  sum := 0;
  For i := 1 to 100000 do begin
    sum := sum + i;  //累计汇总结果
    Position := i;   //记录目前循环的位置
    MyEdit.Text := FloatToStr(sum);
    Synchronize(UpdateCaption); //运行同步更新
    if Terminated then Exit;  //假如收到终止命令则结束
  end;
end;
```

步骤 9 到目前为止，我们已经完成多线程对象的相关声明工作。接下来，我们通过不同的参数类别，调用 TMyFirstThread 对象对应的构造函数，以产生不同显示方式的线程对象。

```
procedure TForm1.ButtonRUN1Click(Sender: TObject);
begin
  //创建第一种线程对象
  TMyFirstThread.Create(Edit1);
end;

procedure TForm1.ButtonRUN2Click(Sender: TObject);
begin
  //创建第二种线程对象
  TMyFirstThread.Create(ProgressBar1);
end;

procedure TForm1.ButtonRUN3Click(Sender: TObject);
begin
  //创建第三种线程对象
  TMyFirstThread.Create(Gauge1);
end;
```

步骤 10 保存并按【F9】键运行工程。工程运行后，您可以同时启动这三个线程对象，这些多线程对象也会分别将其运算结果或进度显示在适当的对象上，如图 21.5 所示。

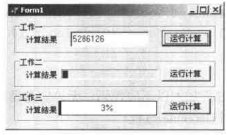


图 21.5 多个线程示范

21.3 多线程与文件复制

Delphi 6 的多线程对象可运用于大部分的情况，笔者无法一一的举列说明。在这个小节中，笔者将示范如何将多线程应用于“文件复制”的应用程序。

范例 21-5

目标：本范例说明如何将多线程应用于“文件复制”的应用程序。

1. 创建新的工程。
2. 在 TForm1 对象中加入两个 TLabel 对象、两个 TEdit 对象、两个 TSpeedButton 对象及一个 TButton 对象，其相关属性设置如表 21.4 所示。

表 21.4

对象	属性名称	属性设置值
TLabel	Name	LabelSource
	Caption	源目录
TLabel	Name	LabelTarget
	Caption	目标目录
TEdit	Name	EditSource
TEdit	Name	EditTarget
TSpeedButton	Name	SpeedButtonSource
TSpeedButton	Name	SpeedButtonTarget
TButton	Name	ButtonRUN
	Caption	运行

3. 从“File/New/Other...”下拉式菜单中打开“New Items”对话框，并从“New”页标签中点选“Thread Object”。
4. 请在“New Thread Object”对话框中输入指定的线程名称，并单击“OK”按钮，这里以 TMyFirstThread 为例。此时系统自动产生一个程序单元文件，并在该单元文件中声明一个名称为 TMyFirstThread 的线程对象。
5. 在 TMyFirstThread 对象中声明必要的变量与子程序，如下所示。

```
type
  TMyFirstThread = class(TThread)
  private
    TotalSize : Integer;  //记录所有被复制的文件大小
    CurrentFile, SourcePath, TargetPath:String;
    //负责线程的更新操作
    procedure UpdateCaption;
    //计算所有被复制的文件大小
    procedure CountFileSize(Path:String);
    //进行文件复制
    procedure myCopyFile(SourcePath,TargetPath:String);
    //获取指定目录下的所有文件
    procedure GetAllFiles(SourcePath,TargetPath:String);
  protected
    procedure Execute; override;
  public
    //改写线程的构造函数
  Constructor Create(v_SourcePath,v_TargetPath:String); overload;
  end;
```

步骤 6　为 CountFileSize 过程内容加入以下的代码，这个过程主要通过递归的方式计算出所有被复制文件的大小。

```
procedure TMyFirstThread.CountFileSize(Path:String);
var
  sr: TSearchRec;
begin
  //Path 变量必须以 "\" 符号结尾
  Path := IncludeTrailingBackslash(path);
  if FindFirst(Path + '\*.*' , faAnyFile, sr) = 0 then begin
     repeat
       if (sr.Name <> '.') and (sr.Name <> '..') then begin
         if sr.Attr <> faDirectory then begin
           TotalSize := TotalSize  + sr.Size;
         end else begin
           CountFileSize(Path + sr.Name);
         end;
       end;
     until FindNext(sr) <> 0;
   end;
end;
```

步骤 7　为 GetAllFiles 过程内容加入以下的代码，这个过程主要通过递归的方式获取所有被复制文件的文件名称，并调用 myCopyFile 过程进行文件的复制工作。

```
procedure TMyFirstThread.GetAllFiles(SourcePath,TargetPath:String);
```

```
var
  sr: TSearchRec;
begin
  if Terminated then Exit;  //假如收到终止命令则结束
  //路径必须以 "\" 符号结尾
  SourcePath := IncludeTrailingBackslash(SourcePath);
  TargetPath := IncludeTrailingBackslash(TargetPath);
  if FindFirst(SourcePath + '\*.*' , faAnyFile, sr) = 0 then begin
    repeat
      if (sr.Name <> '.') and (sr.Name <> '..') then begin
        if sr.Attr <> faDirectory then begin
          myCopyFile(SourcePath+sr.Name,TargetPath+sr.Name);
        end else begin
          GetAllFiles(SourcePath +sr.Name,TargetPath+sr.Name)
        end;
      end;
    until FindNext(sr) <> 0;
  end;
end;
```

8 为 myCopyFile 过程内容加入以下的代码，这个过程主要用来复制指定的单一文件。

```
procedure TMyFirstThread.myCopyFile(SourcePath,TargetPath:String);
var
  SourceFile,TargetFile : File;
  SourceCount, TargetCount : Integer;
  Buffer: array[1..2048] of Char;
begin
  //假如目标目录不存在，先行创建
  ForceDirectories(ExtractFileDir(TargetPath));
  //指定 "源" 与 "目标" 文件
  AssignFile(SourceFile, SourcePath);
  Reset(SourceFile, 1);
  AssignFile(TargetFile, TargetPath);
  Rewrite(TargetFile, 1);
  try
    //每次读取 Buffer 的大小，以进行复制工作
    repeat
      BlockRead(SourceFile,Buffer,SizeOf(Buffer), SourceCount);
      BlockWrite(TargetFile, Buffer, SourceCount, TargetCount);
      CurrentFile :=  SourcePath;
      if Terminated then Exit;  //假如收到终止命令则结束
      synchronize(UpdateCaption);  //更新线程的运算结果
    until (SourceCount = 0) or (TargetCount <> SourceCount);
```

```
  finally
    //结束工作
    CloseFile(SourceFile);
    CloseFile(TargetFile);
  end;
end;
```

Step 9 为 UpdateCaption 过程内容加入以下的代码，这个过程主要用来显示文件复制的进度。这些对象都被定义于 Form3 对象，有关 Form3 对象的相关设置，稍后的步骤将会有详细的介绍说明。

```
procedure TMyFirstThread.UpdateCaption;
begin
   Form3.LabelCurrentFile.Caption := CurrentFile ;
   Form3.ProgressBar1.Max := TotalSize;
   Form3.ProgressBar1.StepBy(2048);
end;
```

Step 10 为线程对象的构造函数内容加入以下的代码。这段代码主要是设置文件复制的“源”与“目标”目录及获取源目录的所有文件大小。

```
Constructor TMyFirstThread.Create(v_SourcePath,v_TargetPath:String);
begin
    inherited Create(False);
    //决定释放的方式
    FreeOnTerminate:=True;
    //获取“源”与“目标”目录
    SourcePath := v_SourcePath;
    TargetPath := v_TargetPath;
    //计算源目录中的所有文件大小
    TotalSize := 0;
    CountFileSize(v_SourcePath);
end;
```

Step 11 为线程对象的 Execute 过程加入以下的代码。这段代码主要是调用 GetAllFiles 子程序以便进行文件复制的操作。

```
procedure TMyFirstThread.Execute;
begin
  GetAllFiles(SourcePath,TargetPath);
end;
```

Step 12 到目前为止，我们已经完成多线程对象的相关声明操作。接下来笔者将创建一个新的 TForm 对象，它将被用来显示复制工作的进度，也就是在步骤 9 所调用的 Form3 对象。请从“File/New/Form”下拉式菜单为此工程加入一个新的 TForm 对象，其默认名称为 Form3 对象。

Step 13 请将 Form3 对象调整至适当大小，并为其加入一个 TAnimate 对象、一个 TProgressBar

对象、一个 TButton 对象及两个 TLabel 对象，其相关属性设置如表 21.5 所示。

表 21.5

对象	属性名称	属性设置值
TAnimate	Name	Animate1
	CommonAVI	aviCopyFiles
TProgressBar	Name	ProgressBar1
TButton	Name	Button1
	Caption	取消
TLabel	Name	Label1
	Caption	正在复制文件
TLabel	Name	LabelCurrentFile

STEP 14 请为 Form3 中 Button1 对象的 onClick 事件处理过程加入以下的代码，这段代码主要是询问用户是否确定终止线程的运行。

```
procedure TForm3.Button1Click(Sender: TObject);
begin
  //先暂停线程
  Form1.Thread1.Suspend;
  if MessageDlg('您确定要停止目前的复制工作吗?'
      ,mtConfirmation,[mbYES,mbNO],0)=mrYES then begin
    Form1.Thread1.Resume; //继续线程的运行
    Form1.Thread1.Terminate; //结束线程
  end else begin
    Form1.Thread1.Resume; //继续线程的运行
  end;
end;
```

STEP 15 到目前为止，我们已经完成了多线程对象与信息对话框的相关设置工作。接下来笔者将示范如何调用文件复制的多线程对象，请在 Form1 对象中声明一个线程变量与相关过程。

```
  TForm1 = class(TForm)
{略}
public
    Thread1 : TMyFirstThread;   //线程变量
    //声明一个过程，当线程结束时会调用此函数
    procedure Thread1Done(Sender : TObject);
  end;
```

STEP 16 为 ThreadDone 过程加入以下的代码，这段代码主要是隐藏信息对话框。

```
procedure TForm1.Thread1Done(Sender : TObject);
begin
  //隐藏信息对话框
```

```
    Form3.Hide;
    ButtonRUN.Enabled := True;
  end;
```

(17) 分别为 Form1 两个 TSpeedButton 对象的 onClick 事件处理过程加入以下的代码，这段代码主要以两种不同的方式打开不同的目录选择对话框，以获取文件复制的“源”与“目标”目录。

```
procedure TForm1.SpeedButtonSourceClick(Sender: TObject);
var
  Title, Root, Dir : String;
begin
  //选择文件复制的“源”目录(uses FileCtrl);
  Title := 'Select Directory';
  Root := ''; //注意 Root 参数，空值代表为最上层
  Dir := '';
  if SelectDirectory(Title,Root,Dir) then EditSource.Text := Dir;
end;

procedure TForm1.SpeedButtonTargetClick(Sender: TObject);
var
  Dir: string;
begin
  //选择文件复制的“目标”目录(uses FileCtrl);
  Dir := 'C:\'; //Dir 为空白代表当前的目录
  if SelectDirectory(Dir, [sdAllowCreate, sdPerformCreate,
     sdPrompt],0) then  EditTarget.Text := Dir;
end;
```

(18) 为 Form1 中 TButton 对象的 onClick 事件处理过程加入以下的代码，这段代码主要是显示信息对话框并通过线程对象进行文件复制的工作。

```
procedure TForm1.ButtonRUNClick(Sender: TObject);
var
  SourcePath,TargetPath:String;
begin
  //显示信息对话框
  Form3.Show;
  Form3.Animate1.Active := True;
  Form3.ProgressBar1.Position := 0;
  Form3.Refresh;
  //获取“源”与“目标”目录
  SourcePath := EditSource.Text;
  TargetPath := EditTarget.Text;
  //运行线程
  Thread1 := TMyFirstThread.Create (SourcePath,TargetPath);
```

```
    Thread1.OnTerminate:=Thread1Done;
    ButtonRUN.Enabled := False;
end;
```

步骤 19 保存并按【F9】键运行工程。工程运行后，请选择文件复制的“源”与“目标”目录并单击“运行”按钮。此时，程序通过信息对话框显示文件复制的进度及被复制的文件名称，如图 21.6 所示。



图 21.6 多线程与文件复制范例

21.4 多线程与数据库应用程序

本小节中，笔者将以实例为您示范如何将多线程应用于数据库应用程序。

范例 21-6

目标：本范例将示范如何将多线程应用于数据库应用程序。

步骤 1 创建新的工程。

步骤 2 在 TForm1 对象中加入两个 TLabel 对象、两个 TEdit 对象、两个 TSpeedButton 对象及一个 TButton 对象，其相关属性设置如表 21.6 所示。

表 21.6

对象	属性名称	属性设置值
TADOConnection	Name	ADOConnection1
	ConnectionString	(请连接到 SQL Server)
	Connected	True
TADODataSet	Name	ADODataSet1
	Connection	ADOConnection1
	CommandText	select * from employee
TADODataSet	Name	ADODataSet2
	Connection	ADOConnection1
	CommandText	select * from jobs

续上表

对象	属性名称	属性设置值
TADODataSet	Name	ADODataSet3
	Connection	ADOConnection1
	CommandText	select * from sales
TDataSource	Name	DataSource1
	DataSet	ADODataSet1
TDataSource	Name	DataSource2
	DataSet	ADODataSet2
TDataSource	Name	DataSource3
	DataSet	ADODataSet3
TDBGrid	Name	DBGrid1
	DataSource	DataSource1
TDBGrid	Name	DBGrid2
	DataSource	DataSource2
TDBGrid	Name	DBGrid3
	DataSource	DataSource3
TButton	Name	Button1

STEP 3 从“File/New/Other...”下拉式菜单中打开“New Items”对话框，并从“New”页标签中点选“Thread Object”。

STEP 4 请在“New Thread Object”对话框中输入指定的线程名称，并单击“OK”按钮，这里以TMyFirstThread为例。此时系统自动产生一个程序单元文件，并在该单元文件中声明一个名称为TMyFirstThread的线程对象。

STEP 5 在TMyFirstThread对象中声明必要的变量与子程序，如下所示：

```
type
  TMyFirstThread = class(TThread)
  private
    myDataSet : TADODataSet; //声明一个 ADO DataSet 变量
  protected
    procedure Execute; override;
  public
    //改变线程的构造函数
    Constructor Create(varDataSet:TADODataSet); overload;
  end;
```

STEP 6 为线程对象的构造函数内容加入以下的代码。这段代码主要是设置线程所使用的TADODataSet对象，并且关闭指定的TADODataSet对象。

```
Constructor TMyFirstThread.Create(varDataSet:TADODataSet);
begin
```

```
    inherited Create(False);
    myDataSet:=varDataSet; //设置 DataSet 对象
    myDataSet.Close;       //先关闭指定的 DataSet
    FreeOnTerminate:=True;
end;
```

7 为线程对象的 Execute 过程内容加入以下的代码。这段代码主要是打开指定的 TADODataSet 对象。

```
procedure TMyFirstThread.Execute;
begin
   myDataSet.Open;  //打开指定的 DataSet
   if Terminated then Exit; //假如收到终止命令则结束
end;
```

8 到目前为止，我们已经完成多线程对象的相关声明工作。接下来笔者将通过 TButton 对象的 onClick 事件处理过程，为三个 TADODataSet 对象分别创建线程以获取数据。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   //为三个 DataSet 分别创建线程以获取数据
   TMyFirstThread.Create(ADODataSet1);
   TMyFirstThread.Create(ADODataSet2);
   TMyFirstThread.Create(ADODataSet3);
end;
```

9 保存并按【F9】键运行工程。工程运行后，当您单击 Button1 按钮时，程序会创建三个线程对象同时进行数据获取的工作，如图 21.7 所示。

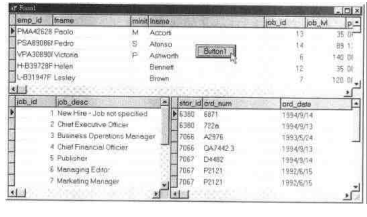


图 21.7 多线程与数据库应用范例

第 22 章

编写 Delphi 6 VCL 组件

本章目标

Delphi 是一个以组件为架构且能快速创建应用程序的开发环境(RAD, Rapid Application Development), 所以 VCL(Visual Component Library)正是协助您快速创建应用程序的重要功臣。本章将示范如何编写属于您自己的 VCL 组件, 通过控制组件的制作, 以协助您更快速地创建各种不同类型的应用程序。本章所含盖的主题包括: 如何创建组件的属性(Property)、操作方法(Method)、事件(Event)及 VCL 组件制作实例等。

本章主题

22.1 如何编写 Delphi VCL 组件
22.2 加上组件面板上的图形文件
22.3 创建组件的属性(Property)
22.4 创建组件的操作方法(Method)
22.5 创建组件的事件(Event)
22.6 VCL 组件制作实例

22.1 如何编写 Delphi VCL 组件

TObject 类别是 Delphi 中所有对象的祖先，而 TComponent 类别则是每一个 VCL(Visual Component Library,可视化组件函数库)组件的共同祖先。图 22.1 将以一个简单的 VCL 类别结构来说明并协助您从现有类别中派生出新的自定义类。

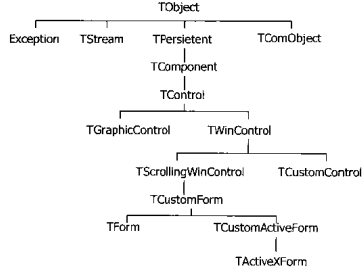


图 22.1 Delphi VCL 的类别架构

从图 22.1 中您可以看出 Delphi 相关 VCL 组件的简单继承结构，以我们最熟悉的 TForm 对象为例。它是从 TObject、TPersientent、TComponent、TControl、TWinControl、TScrollingWinControl、TCustomForm 等类别一路继承下来。因此当您要创建新 VCL 组件时，您可以选择一个最接近的 VCL 组件加以继承，无需从 TObject 开始编写。如果您想查询其他 VCL 组件的继承结构，可以通过联机帮助文件上的 Hierarchy 功能查询即可。图 22.2 示范如何找出 TFrom VCL 的继承结构。

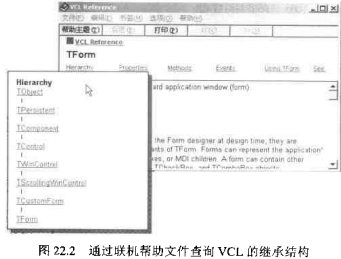


图 22.2 通过联机帮助文件查询 VCL 的继承结构

Delphi 允许您使用下列各种方式来创建 VCL 组件。

- 修改现有的组件。例如:TButton、TEdit、TLabel 或是 TListBox 等。当然，您也可以从这些组件的派生类来创建组件。其对应的派生类为 TCustomButton、TCustomEdit、TCustomLabel 或 Tcustom ListBox 等。
- 创建窗口型的控制组件。创建窗口型的控制组件，您可以继承自 TWinControl 组件。
- 创建图形控制组件。创建图形控制组件，您可以继承自 TGraphic Control 组件。
- 创建非可视化的组件。创建非可视化组件，您可以继承自 TComponent 组件。



22-1

目的：本范例将通过实例的方式为您说明如何创建一个简单的窗口型控制组件。

步骤 1 打开 Delphi，并关闭所有的工程。

步骤 2 从“Component / New Component”下拉式菜单打开“New Component”对话框。其中 Ancestor type 代表所要继承的类；Class Name 代表新的类名称；Palette Page 代表这个组件将来被安装时，它是属于组件面板上的哪一个分类，您也可以为这个组件创建一个新的分类；Unit file name 代表您所要保存的程序单元名称与路径，如图 22.3 所示。

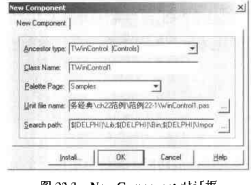


图 22.3　New Component 对话框

步骤 3 在“New Component”对话框中，Search Path 代表 Delphi 的搜索路径，通常在您指定 Unit file name 后，如果它是一个新的路径，那么系统会自动加入，如图 22.4 所示。

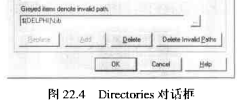


图 22.4　Directories 对话框

STEP 4 如果您想为此组件创建一个 Package 以便在 Delphi 的 IDE 集成环境中安装它的话，那么请单击“Install”按钮。此时出现一个“Install”的对话框。在这个对话中，您可以选择一个已经存在的 Package 或是创建一个新的 Package。笔者这里以创建新的 Package 为例，如图 22.5 所示。

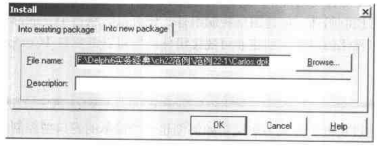


图 22.5 选择一个新的 Package 文件名称

STEP 5 最后，请在“New Component”对话框中单击“OK”按钮。事实上，在这个时候我们已经创建了一个简单的窗口型控制组件了。因此 Delphi 的 IDE 环境会自动进行编译，并产生一个.bpl 的文件。.bpl 文件是一种类似.dll 的文件，它默认被保存在“C:\Program Files\Borland\Delphi6\Projects\Bpl”目录中。此时，系统询问我们是否在编译的同时要顺便安装这个 VCL 组件呢?这里请单击“Yes”按钮，如图 22.6 所示。

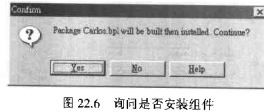


图 22.6 询问是否安装组件

STEP 6 假如系统编译且安装此组件成功后，系统会出现一个确认的对话框。由此您可得知哪些组件已经被安装，这里只有 WinControl1 组件，它是继承自 TWinControl1 类，如图 22.7 所示。

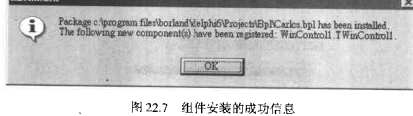


图 22.7 组件安装的成功信息

STEP 7 您可以通过组件面板来查看我们所创建的组件是否加入成功，如图 22.8 所示。

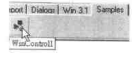


图 22.8 从组件面板中查看组件

步骤 8 请别忘了保存我们所创建的组件。

如果您完成以上的步骤后，系统至少为您产生下列各种文件。

- .dpk 是 Package 的源文件，本范例所产生的是 carlos.dpk。
- .res 是 Package 的资源文件，本范例所产生的是 carlos.res。
- .dof 是 Package 的各种编译器设置文件，本范例所产生的是 carlos.dof。
- .dcu 是 Package 编译后的文件，本范例所产生的是 carlos.dcu。
- .cfg 是 Package configuration 的信息，本范例所产生的是 carlos.cfg。
- .bpl 是 Package 被编译后的一种类似.dll 的文件，本范例所产生的是 carlos.bpl。

上述的六种文件都是与 Package 相关的，因此如果您的组件并没有产生 Package，那么这几个文件应该是不存在的。下面所要介绍的两种文件则是直接与组件有关的文件。

- .pas 是真正保存 VCL 组件 Object Pascal 代码的文件，本范例所产生的是 WinControl1.pas。
- .dcu 是 VCL 组件代码经过编译后的文件，本范例所产生的是 WinControl1.dcu。

接着，笔者将通过 WinControl1.pas 为您介绍创建一个文件所需要的程序架构，以下为 WinControl1.pas 文件的完整内容。

```
unit WinControl1; //程序单元文件名称
interface
uses  //系统自动引用与窗口型控制组件相关的程序单元文件
  Windows, Messages, SysUtils, Classes, Controls;
type //声明一个新的组件类，其继承自 TWinControl
  TWinControl1 = class(TWinControl)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;
//声明一个注册组件的过程
procedure Register;

implementation
//注册组件的过程完整内容
procedure Register;
begin
  //使用 RegisterComponents 程序将此 TWinControl1 组件
  //注册到组件面板上的 Samples 页标签
  RegisterComponents('Samples', [TWinControl1]);
end;
end.
```

这个程序单元文件与其他 TForm 对象的单元文件并没有太大的差异，其最大的差别只有两点：

- 继承的对象不同。我们所熟悉的 TForm 对象单元文件继承自 TForm 对象，而上述的程序单元文件是继承自 TWinControl 类别。
- 访问权的不同。一般的情况下，我们只会针对 TForm 对象的派生类创建 private 与 public 两种访问权。private 代表私有的声明，也就是说，这些声明内容只供 TForm 对象的派生类访问而已；public 则是供所有对象访问，无论是否属于 TForm 对象的派生类的范围。组件的访问权除了 private 与 public 两种外，另外提供 protected、automated 与 published 二种访问权限。Protected 代表这是一种被保护的代码，只有类别本身或其子类才能访问；automated 与 public 一样可供所有对象访问，并且还会产生 Automation 的类型信息；published 与 public 一样可供所有对象访问，并且能将其显示在对象检查器中，我们通常会将组件属性(Property)与事件(Event)的声明放置于此。
- 组件注册过程。Register 是一个组件制作的特殊过程，当组件被安装时，Register 过程必定会被运行，该过程内容记录着与组件注册有关的信息。例如：所属类及组件面板的页标签名称。

现在您已经知道如何创建一个简单的窗口控制组件，所以笔者将为您示范如何使用此组件。

范例 22-2

目的：示范如何使用范例 22-1 所创建的组件。

步骤 1 创建新的工程。

步骤 2 位于组件面板上的“Sample”页标签将上一个范例所创建的组件加入 TForm 对象。TWinControl1 组件被加入后，系统会自动将其命名为 WinControl11，您可以从对象检查器中浏览 WinControl11 对象所提供的属性与事件。WinControl11 对象目前并没有提供任何的事件处理功能，而其所提供的属性如图 22.9 所示。



图 22.9 TWinControl1 组件所默认的属性

步骤 3 保存工程并按【F9】运行工程。工程运行后，您可能会发现根本看不到 WinControl11 对象。没错，那么因为该对象与 TForm 对象颜色相同的缘故。

我们可以查看 Unit1 程序单元文件的内容来证明我们已经将 TWinControl1 组件加入 TForm 对象中。

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, WinControl1; //TWinControl1 所对应的程序单元文件已被自动的加载
type
  TForm1 = class(TForm)
    WinControl11: TWinControl1; //TWinControl1 已被加入 TForm1 对象中
```

22.2　加在组件面板上的图形文件

上一小节中，我们示范如何制作一个简单的窗口型控制组件。但是该组件被加入 Delphi 组件面板中的显示图形为，这是 Delphi 所默认的组件图标。那么要如何更改这个组件图标呢?以下将通过实例的方式示范如何为指定组件加入自定义的组件面板图标。

范例 22-3

目的：示范如何为范例 22-1 所创建的组件加入新的组件面板图标。

步骤 1 从 Delphi 的 IDE 下拉式菜单中选择“Tools/Image Editor”，打开 Delphi 所提供的 Image Editor 图标制作工具。

步骤 2 从下拉式菜单选择“File/New…/Component Resource File(.dcr)”功能以创建一个.DCR 的动态组件资源文件(Dynamic Component Resource)，此时系统默认的文件名称为“Untitled1.dcr”，如图 22.10 所示。

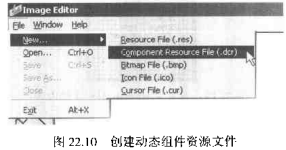


图 22.10　创建动态组件资源文件

步骤 3 在“Untitled1.dcr”中按下右键选择“New/Bitmap”功能以创建一个 Bitmap 图形文件，如图 22.11 所示。

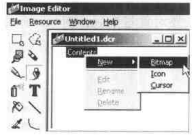


图 22.11 创建 Bitmap 图形文件

STEP 4 系统所默认的图形文件大小为 32*32，请将其调整为 24×24 像素。除此之外，您也可以改变其默认的颜色种类，笔者这里以 VGA 的 16 色为例，如图 22.12 所示。

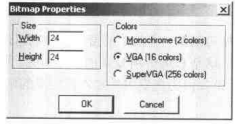


图 22.12 设置 Bitmap 图形文件相关属性

STEP 5 重新为此图形文件命名。这里的命名规则有一些技巧，您必须特别注意。这里的名称必须与组件的类别相同，在范例 22-1 中我们创建一个名称为 TWinControl1 类别的组件。因此这里的图形文件就必须命名为 TWINCONTROL1(请使用大写字母)。

STEP 6 您可以通过右键功能的 Edit 选项进行图形文件的编辑，这里以一个中文字“霖”为例，如图 22.13 所示。

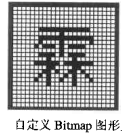


图 22.13 自定义 Bitmap 图形文件内容

STEP 7 到目前为止图形文件的制作算是告一段落，请将此资源文件另存新文件为 WinControl1.dcr，而且必须与 WinControl1.dcu 保存在相同的目录。您可能会问为什么要使用 WinControl1.dcr 这个名字呢?这是因为系统规定.dcr 文件必须与程序单元文件的名称相同，如果您使用{$R *.dcr}来说明资源文件位置的话。

STEP 8 图形文件制作完成后，接下来的工作就是让您的组件认识这个图形文件。我们可以通过{$R WinControl1.dcr}或{$R *.dcr}来说明与此组件有关的资源文件，您可以将这样的语句放在 Package 中，也可以将它放在程序单元文件中。笔者建议您将这样的语句放在程序单元文件中，因为这个资源文件必须伴随着程序单元文件，

如图 22.14 所示。



图 22.14　动态组件资源文件的引用方式

(9) 引用资源文件后，您只要再打开 Carlos.dpk 并单击“Compile”按钮系统会自动为您更新组件面板上的图标，如图 22.15 所示。

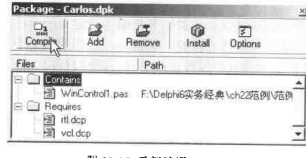


图 22.15　重新编译 Package

(10) 您可以再一次的查看组件面板上的图标是否如图 22.16 所示。

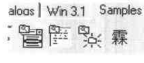


图 22.16　重新查看组件图标

注意　如果您想移除已安装的组件，只要从 Delphi 的 IDE 中选择“Component/Install Packages...”功能，打开“Project Options”对话框，再删除指定的“Package”选项即可，如图 22.17 所示。



图 22.17　删除已安装的组件

22.3 创建组件属性(Property)

属性(Property)对于组件用户而言，它就像是一个变量而且是一个在设计时期就可以通过对象检查器设置与改变的变量。因为它可以在对象检查器中设置属性值，因此它也可以在设计时期就能验证数据的正确性。另外，属性可以隐藏实现的细节，就像 TTable 的 Active 属性一样。我们只要将该属性值设为 True，组件的内部则进行一段非常复杂的运算并且从数据库获取指定 Table 或是 View 的内容。事实上，这也说明了属性的另一个特性，那就是属性可以是虚拟的。因为属性它可能只是一个接口而已，而且无需对应到组件内部的单一字段(Field)值。

22.3.1 公开派生类的属性

大部分的情况我们并不会从 TObject 类别开始编写自定义的组件，而是继承一个比较接近的类再添加新的属性、事件及操作方法等。通常，在我们继承某个类时，系统并不会自动公开所有的属性，因此我们必须将这些属性加以公开。在编写代码之前，我们先以 TWinControl 类为例，比较其父类属性与派生类所自动公开的属性。



（派生类别的公开属性） （TWinControl 类别的属性列表）

图 22.18 派生类与父类的属性对照

从图 22.18 的比较中您可以发现，事实上，系统并不会为派生类公开所有的属性，例如：AlignDisabled、BevelEdges、BevelInner、BevelKind、BevelOuter、BevelWidth、BorderWidth 及 Brush 等。笔者将在下面的范例为您示范如何公开这些属性。

范例 22-4

目的：示范如何为范例 22-3 所创建的组件公开其派生类的属性。

(1) 在 Delphi 的 IDE 环境中打开范例 22-3 所创建的 Package 文件(Carlos.dpk)并打开该组件所属的程序单元文件(WinControl1)。

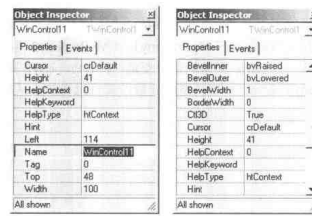
(2) 在 TWinControl1 的 published 模块中加入下列的代码，这段代码主要目的是公开派

生类已经存在的属性包括：Ctl3D、BevelInner、BevelOuter、BevelWidth 及 BorderWidth 等。

```
TWinControl1 = class(TWinControl)
{略}
 published
   { Published declarations }
   //公开派生类已经存在的属性
   property Ctl3D;
   property BevelInner;
   property BevelOuter;
   property BevelWidth;
   property BorderWidth;
 end;
```

3 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

我们可以重新打开范例 22-2 所创建的工程以查看 TWinControl1 的组件内容是否已经更新，如图 22.19 所示。



（属性更新前）　　（属性更新后）

图 22.19　查看组件内容是否被更改

22.3.2　添加直接访问属性

本小节将为您示范如何添加直接访问的属性，这里所谓的直接访问是代表属性的访问并没有经过其他复杂的计算，而是直接访问其所对应的变量值。基本上组件的属性可以是任何数据类型，例如：简单类型(Simple Types)、字符串类型(String Types)、结构类型 (Structured Types)、指针类型 (Pointer Types)及变量类型(Variant Types)等。对象检查器会针对不同的数据类型进行不同的数据验证，例如变量的范围等。以下针对几个常用的属性类型加以说明对象检查器的对应处理方式。

简单类型(Simple Types)：对象检查器允许组件用户直接编辑属性的值，而只进行必要的数据类型检查，例如 Byte 类型只允许 0~255 的整数值等。

- 枚举类型(Enumerated Types)：对象检查器也允许组件用户直接编辑属性的值，也可以通过类似 TListBox 组件的操作方法来显示属性所有的可能值。
- 集合类型(Set Types)：对象检查器也允许组件用户直接编辑属性的值，但只能选择 True 或 False。TDBGrid 组件的 Options 属性就是一个典型的例子。
- 对象类型(Object Types)：通常您必须为此属性设置其专属的编辑器。当组件用户快按属性字段两下时系统就会显示专属编辑器，但是这样的属性必须继承自 TPersistent 类。
- 数组类型(Array Types)：与对象类型的属性一样，系统并没有提供对应的内建编辑器，您必须在组件注册的同时指定其专属的编辑器。

以下将通过实例的演练来示范如何为组件添加直接访问的属性。

范例 22-5

目的：示范如何为范例 22-1 所创建的组件添加直接访问的属性。

Step 1 在 Delphi 的 IDE 环境中打开范例 22-3 所创建的 Package 文件(Carlos.dpk)，并打开该组件所属的程序单元文件(WinControl1)。

Step 2 每一个直接访问的属性必须对应到组件的一个真实变量，因此请在 private 模块中分别创建不同数据类型的变量 FMyProperty1、FMyProperty2 与 FMyProperty3。其中 FMyProperty1 是一个 Byte 类型的整数变量；FMyProperty2 则是一个字符串类型变量；FMyProperty3 是一个布尔类型的变量，其只允许 True 与 False 两种布尔值。

```
TWinControl1 = class(TWinControl)
private
  { Private declarations }
  FMyProperty1 : Byte;
  FMyProperty2 : String;
  FMyProperty3 : Boolean;
```

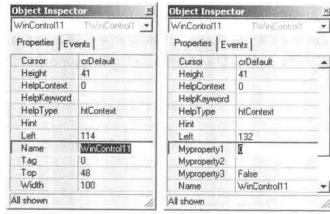
Step 3 在 published 模块中创建三个对应的属性，分别为 MyProperty1、MyProperty2 及 MyProperty3 等。这些属性分别访问 FMyProperty1、FMyProperty2 及 FMyProperty3 等变量。其中 read 代表组件属性被读取时所对应的变量名称；write 代表组件属性被设置时所对应的变量名称。

```
TWinControl1 = class(TWinControl)
(略)
  published
    { Published declarations }
    //添加的"直接访问"属性
    property MyProperty1 : Byte read  FMyProperty1
                                      write FMyProperty1;
    property MyProperty2 : String read  FMyProperty2
                                      write FMyProperty2;
    property MyProperty3 : Boolean read  FMyProperty3
```

```
                                    write FMyProperty3;
  end;
```

步骤 4　您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

我们可以重新打开范例 22-2 所创建的工程以查看 TWinControl1 组件的内容是否已经被更新，如图 22.20 所示。



（属性更新前）　　（属性更新后）

图 22.20　查看组件内容是否被修改

从图 22.20 中您可以发现系统会自动为 MyProperty1 与 MyProperty3 设置初值。除此之外，组件可以在设计时期进行必要的类型检查或范围值的判断。例如图 22.21 就是一个整数类型的检查结果，而图 22.22 则是一个 Byte 类型的范围检查。

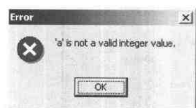


图 22.21　整数类型的检查结果

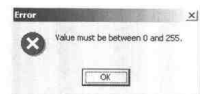


图 22.22　验证属性值的范围是否合法

22.3.3　设置属性默认值

本小节将延续范例 22-5 为您示范如何为属性设置默认值。本小节将为您示范如何通过组件构造函数(Constructor)来设置属性的默认值。

范例 22-6

目的：示范如何为范例 22-5 所创建的组件设置属性的默认值。

步骤 1　在 Delphi 的 IDE 环境中打开范例 22-5 所创建的 Package 文件(Carlos.dpk)，并打开该组件所属的程序单元文件(WinControl1)。

(2) 在 public 模块中声明一个组件构造函数(Constructor)。因为这算是改写系统所默认的构造函数，因此必须加上 override 保留字。

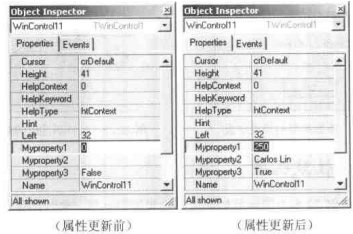
```
TWinControl1 = class(TWinControl)
(略)
  public
    //创建组件的构造函数 Constructor
    Constructor Create(AOwner:TComponent);override;
```

(3) 在 implementation 模块中定义自定义的构造函数内容，这段程序首先继承自系统默认的构造函数功能，并且设置 FMyproperty1、Fmyproperty2 与 Fmyproperty3 等变量的默认值。

```
Constructor TWinControl1.Create(AOwner:TComponent);
begin
   inherited Create(AOwner); //继承原有构造函数的功能
   //设置属性默认值
   FMyproperty1 := 250;
   FMyproperty2 := 'Carlos Lin';
   FMyproperty3 := True;
end;
```

(4) 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

我们可以重新打开范例 22-2 所创建的工程以查看 TWinControl1 的组件内容是否已经更新，如图 22.23 所示。



(属性更新前)　　(属性更新后)

图 22.23　查看组件内容是否被修改

22.3.4　添加间接访问属性

本小节将延续范例 22-5 继续为您示范如何添加间接访问的属性。所谓间接访问的属性就是在访问属性前会经过运算，它可以是一个简单的运算过程，也可以是一个很复杂的运算。本小节将为您示范如何添加间接访问的属性。



22-7

目的：示范如何将范例 22-5 所创建的组件修改为间接访问的属性，这里以设置属性值为例。

STEP 1 在 Delphi 的 IDE 环境中打开范例 22-5 所创建的 Package 文件(Carlos.dpk)，并打开该组件所属的程序单元文件(WinControl1)。

STEP 2 在 private 模块中声明一个设置 Myproperty1 属性值的过程，其名称为 SetMyproperty1。

```
TWinControl1 = class(TWinControl)
private
(略)
  //声明一个设置 Myproperty1 属性值的过程
  procedure SetMyproperty1(value : Byte);
```

STEP 3 在 implementation 模块中定义 Myproperty1 过程的内容，这段程序将会重新设置 FMyproperty1 的变量值。而 FMyproperty1 变量值将被反应于 Myproperty1 属性。

```
procedure TWinControl1.SetMyproperty1(value : Byte);
begin
  FMyproperty1 := value * 2;
end;
```

STEP 4 重新声明位于 published 模块的 MyPerperty1 属性。

```
property Myproperty1: Byte read FMyproperty1
                                write SetMyproperty1;
```

STEP 5 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

事实上，您无法用目视的方式来判断组件的内容是否改变。我们可以重新打开范例 22-2 所创建的工程并为 TWinControl1 组件的 Myproperty1 属性设置属性值。设置后系统会将用户的设置值乘以 2 再加以显示。

22.3.5　另一种间接访问属性

本小节将通过索引值(Index)为您示范如何创建间接访问的属性。



22-8

目的：示范如何将范例 22-3 所创建的组件修改为间接访问的属性。

STEP 1 在 Delphi 的 IDE 环境中打开范例 22-3 所创建的 Package 文件(Carlos.dpk)并打开该组件所属的程序单元文件(WinControl1)。

STEP 2 分别在 private、protected、public 及 published 模块中声明变量、子程序、并公开三个属性值(Year、Month 及 Day)。

```
TWinControl1 = class(TWinControl)
```

```
  private
    FDate : TDateTime;  //创建一个 TDateTime 变量
  protected
    //分别创建“获取”和“设置”属性值的函数
    function  GetmyDate(Index : Integer):Integer;
    procedure SetmyDate(Index,value : Integer);
  public
    //声明组件的构造函数
    Constructor Create(AOwner:TComponent);override;
  published
    //公开三个属性，其索引值分别为 1、2、3。
    //其属性的“获取”和“设置”都使用相同的函数
    property Year : Integer index 1 read GetmyDate
                                          write SetmyDate;
    property Month : Integer index 2 read GetmyDate
                                          write SetmyDate;
    property Day : Integer index 3 read GermyDate write SetmyDate;
  end;
```

步骤 3 在 implementation 模块中定义自定义的构造函数内容，这段程序首先继承自系统默认的构造函数功能，并且设置 FDate 的变量初始值为今天的日期。FDate 变量的初始值将会反应于对象检查器中组件相关的属性字段。

```
Constructor TWinControl1.Create(AOwner:TComponent);
begin
    inherited Create(AOwner);
    //设置属性默认值为今天的日期
    FDate := Date;
end;
```

步骤 4 在 implementation 模块中定义自定义的 GetmyDate 函数内容，这段程序会根据索引值(Index)的不同而分别返回“年”、“月”或“日”的内容值。

```
function TWinControl1.GetmyDate(Index : Integer):Integer;
var
   aYear, aMonth, aDay : Word;
begin
   //分解 FDate 变量的内容值
   DecodeDate(FDate,aYear, aMonth, aDay);
   Case Index of
     //根据 Index 来决定返回“年”、“月”或“日”
     1 : Result := aYear;
     2 : Result := aMonth;
     3 : Result := aDay;
     else Result := 0;
   end;
```

```
end;
```

5 在 implementation 模块中定义自定义的 SetmyDate 过程内容，这个过程有 Index 和 value 两个参数。前者代表用户将要设置的字段索引，而后者则代表其将要设置的值。

```
procedure TWinControl1.SetmyDate(Index,value : Integer);
var
  aYear, aMonth, aDay : Word;
begin
  //分解 FDate 变量的内容值
  DecodeDate(FDate,aYear, aMonth, aDay);
  Case Index of
    //根据 Index 重新设置 "年" 、 "月" 或 "日" 的内容值
    1 : aYear := Value;
    2 : aMonth := Value;
    3 : aDay := Value;
    else Exit;
  end;
  //重新组合 "年" 、 "月" 或 "日" 并将其传给 FDate 变量
  FDate := EncodeDate(aYear, aMonth, aDay);
end;
```

6 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

我们可以重新打开范例 22-2 所创建的工程以查看 TWinControl1 的组件内容是否已经更新，如图 22.24 所示。



图 22.24　查看组件内容是否被修改

22.4　创建组件的操作方法(Method)

您可以将操作方法(Method)视为组件的一个子程序。声明一个操作方法时必须注意其访问权限，如果您允许其他的程序访问指定的操作方法，那么您必须将此操作方法的访问权限

设为 Public(公有类型)或 Protected(保护类型)，这两者在使用上并没有太大的差异。

范例 22-9

目的：示范如何将为范例 22-3 所创建的组件添加一个操作方法。

STEP 1 在 Delphi 的 IDE 环境中打开范例 22-3 所创建的 Package 文件(Carlos.dpk)并打开该组件所属的程序单元文件(WinControl1)。

STEP 2 在 public 模块中声明两个子程序以做为此组件的操作方法。GetMyDate 函数没有参数，只有一个字符串类型的返回值；GetMyDate2 过程有一个“输出(out)”的参数，负责返回运算的结果。

```
TWinControl1 = class(TWinControl)
(略)
  public
    function GetMyDate:String;
    procedure GetMyDate2(out MyDate:String);
```

STEP 3 在 implementation 模块中定义 GetMyDate 函数与 GetMyDate2 过程的内容，这两个子程序分别以不同的格式返回今天的日期。

```
function TWinControl1.GetMyDate:String;
var
  myDate : String;
begin
  DateTimeToString(myDate,'yyyy/mm/dd',Now);
  Result := myDate;
end;

procedure TWinControl1.GetMyDate2(out MyDate:String);
begin
  DateTimeToString(myDate,'mm/dd/yyyy',Now);
end;
```

STEP 4 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

接着，笔者将以实例的方式来验证范例 22-9 所创建的操作方法。

范例 22-10

目的：示范如何读取范例 22-9 所创建组件的操作方法。

STEP 1 创建新的工程并在 TForm 对象中分别加入一个 TWinControl1 对象、两个 TEdit 对象及一个 TButton 对象。

STEP 2 在 TButton 对象的 onClick 事件处理过程中加入以下的代码，这段代码主要的目是分别调用 TWinControl1 对象的两个操作方法以获取不同格式的今天的日期。

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  MyDate : String;
begin
  //调用 GetMyDate 操作方法
  Edit1.Text := WinControl11.GetMyDate;
  //调用 GetMyDate2 操作方法
  WinControl11.GetMyDate2(MyDate);
  Edit2.Text := MyDate;
end;
```

3 保存并按【F9】键运行工程。工程运行后，您可以单击 TButton 对象以获取两种不同格式的今天的日期，如图 22.25 所示。



图 22.25　操作方法运行范例

22.5　创建组件的事件(Event)

对于组件而言，事件与属性并没有太大的差异，惟一的不同是前者连接到一个特定类的操作方法。一般来说，我们会为自定义组件加上 Windows 的标准事件。只要是继承自 TControl 对象的派生类都继承了 onClick、onDblClick、onDragDrop、onDragOver、onEndDrag、onMouseMove、onMouseDown 及 onMouseUp 等事件。而继承自 TWinControl 的派生类除了继承上述的八个事件外，另外还继承了 onEnter、onKeyUp、onKeyDown、onKeyPress 及 OnExit 等事件。以下笔者将以实例的方式示范如何公开既有的事件与改变标准事件的处理方式。

范例 22-11

目的：示范如何将为范例 22-3 所创建组件公开一个既有的事件。

1 在 Delphi 的 IDE 环境中打开范例 22-3 所创建的 Package 文件(Carlos.dpk)，并打开该组件所属的程序单元文件(WinControl1)。

2 在 published 模块中公开一个既有的事件，这里以 onClick 为例。

```
  TWinControl1 = class(TWinControl)
(略)
  published
    //公开已有的事件
    property onClick;
```

3 您只要重新编译这个 Package，系统将会自动为您更新组件的内容。

我们可以重新打开范例 22-2 所创建的工程以查看 TWinControl1 的组件内容是否已经更新。从下图中，您可以发现 onClick 事件已经被更新于对象检查器，因此您可以与使用其他组件一样来操作此事件处理过程，如图 22.26 所示。

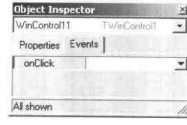


图 22.26 从对象检查器查看添加的事件

22.6 VCL 组件制作实例

笔者将以三个小节的篇幅来讨论几个 VCL 组件制作的实例，这些实例包括：修改既有的 VCL 组件、制作对话框的 VCL 组件及制作访问注册表(Registry)的 VCL 组件等。

22.6.1 修改既有的 VCL 组件

本小节将为您示范如何修改既有的 VCL 组件。这里将以 TCustomLabel 类为例制作一个具有 URL 连接能力的 VCL 组件。



目的：示范如何修改既有的 VCL 组件。

步骤 1 打开 Delphi，并关闭所有的工程。

步骤 2 从“Component / New Component”下拉式菜单打开“New Component”对话框，如图 22.27 所示。

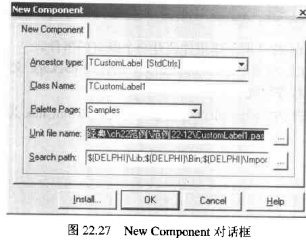


图 22.27 New Component 对话框

注意

其中 Ancestor type 代表所要继承的类，请选择 TCustomLabel 类；Class Name 代表新的类名称，这里以 TMyURLLabel 为例；Palette Page 请使用默认的 Samples 页标签；最后在 Unit file name 字段中选择所要保存的程序单元名称与路径。另外有一件事您必须特别注意，假如您是选择 TCustomLabel [QStdCtrls]而非 TCustomLabel [StdCtrls]类时，在编译的过程中会发生系统所默认引用的程序单元文件并无法顺利通过编译。因此笔者建议您使用 TCustomLabel [StdCtrls]类来创建组件。

3 请单击“Install”按钮为此组件创建一个 Package，以便在 Delphi 的 IDE 集成环境中安装，如图 22.28 所示。

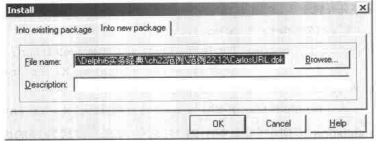


图 22.28　选择一个新的 Package 文件名称

4 请在“New Component”对话框中单击“OK”按钮，并且允许系统在编译的同时安装这个 VCL 组件，如图 22.29 所示。

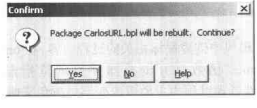


图 22.29　询问是否安装组件

5 在系统安装完成的信息出现后，请查看 MyURLLabel 程序单元文件的内容。这个时候您只得到一个继承自 TCustomLabel 的空组件而已。

```
(略)
type
 TMyURLLabel = class(TCustomLabel)
(略)
```

6 首先在 TMyURLLabel 类型的私有变量声明模块中加入以下的代码。这段代码主要的目的是声明一个 FURL 的变量，其用于保存用户所设置的 URL 内容、声明一个设置 FURL 变量内容的过程，并且声明鼠标的相关事件，包括:onClick、onMouseUp 及 onMouseDown 等。这些事件因为声明于私有变量模块，因此并不会显示于对象检查器中。

```
type
  TMyURLLabel = class(TCustomLabel)
  private
   //声明一个私有变量，其目的在保存 URL 的内容
   FURL:String;
   //声明一个设置 URL 内容的过程
   Procedure SetURL(Value: String);
   //声明鼠标事件，但这些事件并不会出现在对象检查器中
   Property OnClick;
   Property OnMouseUp;
   Property OnMouseDown;
  //声明鼠标的事件处理
   procedure Click(Sender: TObject);
   Procedure MouseUp(Sender: TObject; Button: TMouseButton;
                                  Shift: TShiftState; X, Y: Integer);
   Procedure MouseDown(Sender: TObject; Button:
                  TMouseButton; Shift: TShiftState; X, Y: Integer);
```

STEP 7 在 implementation 模块中创建 SetURL 过程的内容，这段程序的内容主要是将用户所指定的 URL 保存于 FURL 变量中。

```
procedure TMyURLLabel.SetURL(value:string);
begin
  //将使用设置的 URL，保存于 FURL 变量中
  FURL:=value;
end;
```

STEP 8 在 implementation 模块中创建鼠标相关的过程内容。在 onClick 事件处理过程中主要是通过 ShellExecute 函数打开浏览器并且浏览指定的网址；onMouseUp 与 onMouseDonw 事件处理过程中主要以改变字体颜色来响应用户。

```
procedure TMyURLLabel.Click(Sender: TObject);
begin
  //uses ShellApi;
  //使用 ShellExecute 函数打开浏览器并且浏览指定的网址
  ShellExecute(Application.Handle, 'open', pchar(URL), nil, nil ,
                        SW_SHOWNORMAL);
end;

Procedure TMyURLLabel.MouseUp(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
Begin
 //当用户按下鼠标左键则将 TMyURLLabel 的文字设置为蓝色
 If Button = mbLeft Then Font.Color := clBlue;
End;
```

```
Procedure TMyURLLabe .MouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
Begin
 //当用户按下鼠标左键则将 TMyURLLabel 的文字设置为红色
 If Button = mbLeft Then Font.Color := clRed;
End;
```

(9) 在 TMyURLLabel 类型的 published 模块中公开一个既有的 Caption 属性及添加一个 URL 属性。

```
 TMyURLLabel = class(TCustomLabel)
(略)
 published
  //自定义 URL 属性
  property URL : string read FURL write SetURL;
  //显示已经存在的属性
  property Caption;
```

(10) 在 TMyURLLabel 类型的公有变量声明模块中加入以下的代码，这段代码主要是声明一个组件构造函数。

```
 TMyURLLabel = class(TCustomLabel)
(略)
 public
  //组件的构造函数
  Constructor Create(aOwner: TComponent); Override;
```

(11) 在 implementation 模块中创建组件构造函数的内容，这段程序的内容主要是设置相关属性的默认值并指定鼠标事件对应的处理过程。

```
constructor TMyURLLabel.Create;
begin
  inherited Create(aOwner);
  //设置 URL 与 Caption 属性的默认值
  URL := 'http://';
  Caption := 'Carlos Studio';
  //默认字体的设置值
  Font.Size := 14;
  Font.Color := clBlue;
  Font.Style := [fsUnderline];
  //默认鼠标光标
  Cursor := crHandPoint;
  //指定事件所对应的事件处理过程
  OnMouseUp := MouseUp;
  OnMouseDown := MouseDown;
  OnClick := Click;
End;
```

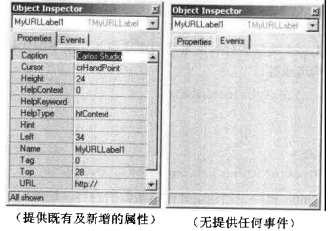
(12) 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

修改既有的组件可以说是创建组件最简单的方法，您可以继承 Delphi 所提供的任何 VCL 组件再派生出属于自己的新组件。继承这些既有 VCL 组件时，我们通常会使用既有的组件或其抽象类。这些抽象类通常以 TCustom 为首，例如：TCustomLabel、TCustomEdit 及 TcustomListBox 等。

范例 22-13

目的：笔者将通过实作的方式来示范如何使用范例 22-12 所创建的 URL 组件。

(1) 创建新的工程并在默认的 TForm 对象中加入范例 22-12 所制作的 TMyURLLabel 组件。您可以在运行时期设置 TMyURLLabel 组件的属性，也可以通过对象检查器设置相关属性，如图 22.30 所示。



（提供既有及新增的属性） （无提供任何事件）

图 22.30 查看新组件的属性与事件处理

注意 虽然在范例 22-12 中，我们编写很多事件处理的代码，为什么在对象检查器中没发现任何的事件呢?那是因为 onClick、onMouseDonw 与 onMouseUp 等事件是被声明于私有(private)模块而非 published 模块，所有无法公开于对象检查器中。

(2) 保存并按【F9】键运行工程。工程运行后，TMyURLLabel 组件可接收 onClick、onMouseDonw 与 onMouseUp 等鼠标事件并且在 onClick 事件处理过程中调用浏览器及浏览指定的网址，如图 22.31 所示。



图 22.31 TMyURLLabel 组件运行范例

22.6.2　制作对话框的 VCL 组件

上一个小节中笔者示范如何修改一个既有的 VCL 组件。事实上，您也可以在 VCL 组件调用其他的对象，例如 TForm 对象或自定义的对话框等。本小节将以实例为您说明如何制作一个“关于”的对话框。



22-14

目的：笔者将以实例的方式来示范如何制作一个“关于”的对话框。

1 参考范例 22-12 制作一个继承自 TComponent 的新组件，这里以 TCarlosAboutForm 为例，并将此组件加入 Package 中。其完整的代码如下：

```
unit CarlosAboutForm;
interface
uses
  Windows, Messages, SysUtils, Classes;
type
  TCarlosAboutForm = class(TComponent)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('Samples', [TCarlosAboutForm]);
end;
end.
```

2 您可以直接在上述的单元文件中声明一个“关于”的对话框，当然也可以使用其他的程序单元文件来声明它。这里我们以 Delphi 的关于样版为例。请从“File/New/Other...”打开“New Items”对话框并且从“Forms”页标签中点选“About Box”，为此组件加入一个“关于”对话框。请将此程序单元文件从 Unit1.pas 另存为其他名称的程序单元文件，这里以 UnitAboutForm.pas 为例。



为什么笔者要特别强调它的命名方式呢?因为笔者在测试的过程中发现，当自定义组件中包括其他名称如 Unit1.pas 的单元文件时，将来在使用上可能会发生不可预期的错误，因为打开 Delphi 的 IDE 环境时其默认的程序单元文件就是 Unit1.pas。

STEP 3 接着，请在 public 模块中声明一个 Execute 的操作方法并且在 implementation 区段中编写其对应的程序内容。这段代码的目的主要是调用步骤 2 所创建的“关于”对话框。

```
TCarlosAboutForm = class(TComponent)
(略)
  public
    function Execute: Boolean; virtual;
(略)
function TCarlosAboutForm.Execute: Boolean;
var
  //uses UnitAboutForm; //引用自定义的程序单元文件
  FMyAbout: TAboutBox;  //声明 TAboutBox 的变量
begin
  //创建关于对话框
  FMyAbout:= TAboutBox.Create(nil);
  try
    FMyAbout.Showmodal; //显示
  finally
    FMyAbout.Free;      //释放
  end;
  Result := True;       //设置返回值
end;
```

STEP 4 您只要重新编译这个 Package 系统将会自动为您更新组件的内容。

如果您所创建的新组件继承自 TComponent 时，则该组件就是所谓的非可视组件(non-visiable)。因此将来在程序运行时是看不见它的，就好像是 TTimer、TTable 或其他的对话框组件一样。



22-15

目的：笔者将通过实作的方式来示范如何使用范例 22-14 所创建的组件。

STEP 1 创建新的工程并在默认的 TForm 对象中加入范例 22-14 所制作的 TCarlosAboutForm 组件。

STEP 2 另外在 TForm 对象中加入一个 TButton 对象并在其 onClick 事件处理程序中调用上一步骤所加入的 TCarlosAboutForm 组件。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //忽略其返回值，直接调用其 Execute 操作方法
```

```
    CarlosAboutForm1.Execute;
end;
```

步骤 3　保存并按【F9】键运行工程。工程运行后当您按下 TButton 对象时将显示组件中所自定义的“关于”对话框，如图 22.32 所示。

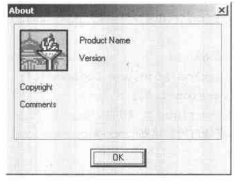


图 22.32　TCarlosAboutForm 组件运行范例

上面的范例示范如何制作一个可以调用其他对话框的组件，但是这个组件还是有些不足。例如：无法设置其中的“产品名称”、“版本”、“版权”及“说明”等字段。事实上，要做到这些功能也不难，您只要参考前面几个小节的实例说明，相信应该不难完成才对。

范例 22-16

目的：示范如何为范例 22-14 所创建的组件加入“产品名称”、“版本”、“版权”及“说明”等字段的修改接口。

步骤 1　打开范例 22-14 的 MyAboutForm.dpk 文件。

步骤 2　在 CarlosAboutForm.pas 中为 TCarlosAboutForm 组件加入四个直接访问的属性。有关直接访问的属性，请参考第 22.3.2 节的详细说明。

```
TCarlosAboutForm= class(TComponent)
private
  FProductName  : String;
  FVersion      : String;
  FCopyright    : String;
  FComments     : String;
{略}
published
  property  ProductName: String read FProductName
                                        write FProductName;
  property Version: String read FVersion  write FVersion;
  property Copyright: String read FCopyright write FCopyright;
  property Comments: String  read FComments write FComments;
```

步骤 3　为 TCarlosAboutForm 组件创建一个构造函数，用来设置该组件的属性默认值。

```
(略)
  public
(略)
    constructor Create(AOwner: TComponent); override;
(略)
constructor TCarlosAboutForm.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    FProductName  := 'Carlos AboutBox Component';
    FVersion      := 'Version 1.0';
    FCopyright    := 'Copyright © 2001-';
    FComments     := '有任何问题请电:xxx-xxx-xxx-xxx';
end;
```

步骤 4 修改 TCarlosAboutForm 组件的 Execute 操作方法，这段代码主要是将“产品名称”、“版本”、“版权”及“说明”等变量内容显示于对话框中。

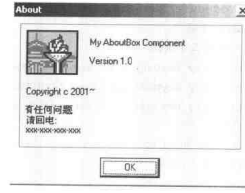
```
function TCarlosAboutForm.Execute: Boolean;
var
  FMyAbout: TAboutBox;  //声明 TAboutBox 的变量
begin
  FMyAbout:=  TAboutBox.Create(nil); //创建关于对话框
  try
    FMyAbout.ProductName.Caption  := FProductName;
    FMyAbout.Version.Caption := FVersion;
    FMyAbout.Copyright.Caption := FCopyright;
    FMyAbout.Comments.Caption := FComments;
    FMyAbout.Showmodal; //显示
  finally
    FMyAbout.Free;      //释放
  end;
  Result := True;       //设置返回值
end;
```

步骤 5 您只要重新编译这个 Package，系统将会自动为您更新组件的内容。

如果您想测试组件是否成功的修改，请您打开范例 22-15 且重新创建 TCarlosAboutForm 对象。此时您可以通过对象检查器更改相关属性值，其运行结果如图 22.33 所示。



（新组件默认的属性值）



（新组件运行结果）

图 22.33　重新查看 TCarlosAboutForm 组件运行结果

22.6.3　制作访问注册表(Registry)的 VCL 组件

在上一个小节的范例中笔者示范如何继承自 TComponent 对象制作一个包含 TForm 对象的关于对话框。这种继承自 TComponent 对象的组件通常没有可见的组件接口，也就是所谓的非可视化组件(non-visible)。这一类的组件就像“Dialogs”页标签中所提供的对话框组件一样，还有其他的数据库组件也都是相同的。笔者在本小节继续为您介绍另两个非可视化组件，这些组件包括“读取 Registry”与“设置 Registry”等。

范例 22-17

目的：示范如何制作一个读取注册表(Registry)的组件。

Step 1　参考上一小节的范例创建一个 TMyRegistryWrite 类的组件，并将其安装于名称为 MyRegistry1.dpk 的 Package。

Step 2　为 TMyRegistryWrite 类创建三个属性及一个操作方法，其完整的代码如下：

```
unit MyRegistryWrite;
interface
uses
  Windows, Messages, SysUtils, Classes,Registry,Dialogs;
type
  TMyRegistryWrite = class(TComponent)
  private
    //设置私有变量
    FKeyPath : String;
    FKeyName : String;
    FKeyValue : String;
  protected
  public
    //声明操作方法
```

```
    procedure SaveKey;
  published
    //公开三个属性
    property KeyPath : string read FKeyPath write FKeyPath;
    property KeyName : string read FKeyName write FKeyName;
    property KeyValue : string read FKeyValue write FKeyValue;
  end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('Samples', [TMyRegistryWrite]);
end;
procedure TMyRegistryWrite.SaveKey;
var
  myRegistry: TRegistry;  //uses Registry
begin
  myRegistry := TRegistry.Create();  //创建 TRegistry 对象
  //设置根目录
  myRegistry.Rootkey:=HKEY_CURRENT_USER;
  myRegistry.OpenKey(FKeyPath, True);    //打开指定的路径
  //设置指定 Key 的内容值
  myRegistry.WriteString(FKeyName,FKeyValue);
  myRegistry.Free;                //释放 TRegistry 对象
  Showmessage('数据已保存');  //use Dialogs
end;
end.
```

在经过编译及安装操作后，您可以打开一个新的工程来测试此组件是否可运作顺利。范例 22-18 将为您示范如何通过范例 22-17 所创建的组件将数据写入指定的路径。

范例 22-18

目的：本范例将示范如何通过范例 22-17 所创建的组件将数据写入指定的路径。

步骤 1 创建新的工程并且为 TForm 对象加入一个 TMyRegistryWrite 组件，其相关属性设置如表 22.1 所示。

表 22.1

对象	属性名称	属性设置值
TMyRegistryWrite	Name	MyRegistryWrite1
	KeyName	Test
	KeyPath	D1\D2\D3\
	KeyValue	12345

步骤 2　为 TForm 对象加入一个 TButton 对象并在 TButton 对象的 onClick 事件处理过程中加入以下代码。这段代码的目的是调用 TMyRegistryWrite 对象的 SaveKey 操作方法以保存指定的设置值。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MyRegistryWrite1.SaveKey;
end;
```

步骤 3　在工程运行后。如果您按下 TButton 对象时，系统会调用 TMyRegistryWrite 对象的 SaveKey 操作方法以便将指定的数据写入注册表。您可以通过注册编辑器来查看它，如图 22.34 所示。

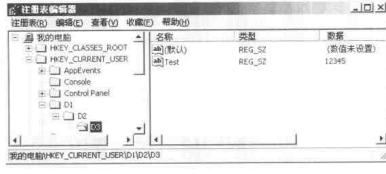


图 22.34　通过注册编辑器查看运行结果

在上面的范例中，笔者已经示范如何将指定数据写入注册表。接下来的范例笔者将为您示范如何读取注册表的内容。

范例 22-19

目的：示范如何制作一个读取注册表(Registry)的组件。

步骤 1　参考范例 22-17 创建一个 TMyRegistryRead 类的组件，并将其安装于名称为 MyRegistry2.dpk 的 Package。

步骤 2　为 TMyRegistryRead 类创建两个属性及一个操作方法，其完整的代码如下：

```
unit MyRegistryRead;
interface
uses
  Windows, Messages, SysUtils, Classes,Registry;
type
  TMyRegistryRead = class(TComponent)
  Private
    //声明两个私有变量
    FKeyPath : String;
    FKeyName : String;
  protected
```

```
  public
    Function GetKey:String; //声明操作方法
  Published
    //公开两个属性
    property KeyPath : string read FKeyPath write FKeyPath;
    property KeyName : string read FKeyName write FKeyName;
  end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('Samples', [TMyRegistryRead]);
end;
Function TMyRegistryRead.GetKey:String;
var
  myRegistry: TRegistry; //uses Registry
  str : String;
begin
  myRegistry := TRegistry.Create(); //创建 TRegistry
  //设置根目录
  myRegistry.Rootkey:=HKEY_CURRENT_USER;
  myRegistry.OpenKey(FKeyPath, True);  //打开指定路径
  Str := myRegistry.ReadString(FKeyName); //读取指定数据
  myRegistry.Free; //释放
  Result := Str;   //返回内容值
end;
end.
```

在经过编译及安装操作后，您可以打开一个新的工程来测试此组件是否能运作顺利。范例 22-20 中将示范如何通过范例 22-19 所创建的组件读取指定路径的数据。



22-20

目的：示范如何使用范例 22-19 所创建的组件读取指定路径的数据。

步骤 1 创建新的工程并且为 TForm 对象加入一个 TMyRegistryWrite 对象、一个 TEdit 对象及一个 TButton 对象，其相关属性设置如表 22.2 所示。

表 22.2

对象	属性名称	属性设置值
TMyRegistryRead	Name	MyRegistryRead1
	KeyName	Test
	KeyPath	D1\D2\D3\

续上表

对象	属性名称	属性设置值
TEdit	Name	Edit1
TButton	Name	Button1

3 在 Button1 对象的 onClick 事件处理过程中加入以下的代码。这段代码主要目的是调用 TMyRegistryRead 对象的 GetKey 操作方法以便获取指定路径的内容值。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := MyRegistryRead1.GetKey;
end;
```

4 在工程运行后。当您按下 TButton 对象时，程序将从注册表中取出指定路径的数据，如图 22.35 所示。

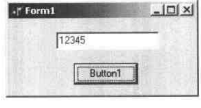


图 22.35　TmyRegistryRead 对象运行结果

附录 A

Object Pascal 程序设计（一）

本章目标

每一种程序语言都有其特殊的语法。笔者认为如果您想学好 Delphi，那么您就必须先熟悉 Object Pascal 的基础语法。为了帮助 Delphi 初学者能快速且有效的入门，笔者特别以两个附录章节来讨论 Object Pascal。本章主要讨论的主题包括：Object Pascal 程序结构、Corsole 应用程序、Windows 应用程序、工程与程序单元文件、程序注释、变量声明、常量声明、语句、过程与函数及程序块等。

本章主题

A.1 Object Pascal 程序结构

A.2 Console 应用程序

A.3 Windows 应用程序

A.4 工程与程序单元文件

A.5 程序注释

A.6 变量声明

A.7 常量声明

A.8 语句(Statement)

A.9 过程(Procedure)与函数(Function)

A.10 程序块

A.1 Object Pascal 程序结构

在讨论 Object Pascal 程序结构之前，我们先来谈谈 Object Pascal 与 Delphi 之间的关系。Delphi 是一个面向对象、可视化的程序设计环境，您可以使用 Delphi 快速开发各种不同类型的应用程序。也就是说 Delphi 是一个以组件为结构且能快速创建应用程序的开发环境(RAD, Rapid Application Development)，而其使用的程序语言源自 Pascal。为了支持 Delphi 的组件结构，因此 Pascal 经过适当修正后的版本就是 Object Pascal。Object Pascal 是一种高级的程序语言，它与 C/C++等程序语言一样需要经过编译及详细的数据类型检查。Object Pascal 的另一个特色是同时支持结构化与面向对象的程序设计方式。所有的高级程序语言都有一个共同的特点，那就是代码的可读性很高。没错，Delphi 除了代码的可读性高以外，另外还允许我们将一个应用程序分为多个程序单元文件以配合模块化程序设计的原则。

如果您使用 Delphi 开发应用程序，那么一个应用程序将会被拆成多个模块。以一个默认的工程而言，它至少被拆为工程文件(.DPR)、程序单元文件(.PAS)、窗体文件(.DFM)、资源文件(.RES)及工程设置文件(.DOF)等。

- 工程文件(.DPR)包含 Delphi 程序的运行顺序及说明窗体文件与程序文件的来源。
- 程序单元文件(.PAS)通常用来记录 Object Pascal 代码。一般来说程序单元文件中包含应用程序大部分的代码，一个应用程序可以包含多个程序单元文件。
- 窗体文件(.DFM)主要记录窗体(TForm)对象上所包含组件的属性及设置信息，一个窗体文件必定会对应一个程序单元文件。
- 资源文件(.RES)用来保存工程中所使用的资源信息，包含工程中所使用的图标(Icon)及图形(Bitmap)等，其文件名称通常与工程名称相同。
- 工程设置文件(.DOF)用来保存工程所使用的各种编译器设置信息，文件名称通常与工程名称相同。

上述的几个模块中与 Object Pascal 有关的只有工程文件(.DPR)与程序单元文件(.PAS)。这些 Object Pascal 代码是通过一个文件头(Heading)标识符来判断其所属的文件类型为何？例如工程文件是由一个“program”标识符为开头，而程序单元文件则是以“Unit”为开头，如图 A.1 所示。

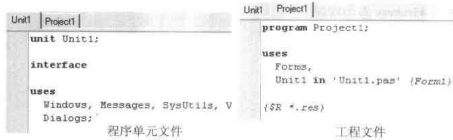


图 A.1

A.2 Console 应用程序

通过上一小节的详细说明，相信您对 Object Pascal 的程序结构应该有了初步的认识。本小节将通过简单 Console 应用程序的编写来说明 Object Pascal 工程文件的最基本结构。您可以通过两种方式来编写 Console 应用程序，第一种方法是使用文本编辑器，例如记事本。第二种方法则是使用 Delphi 的 IDE 开发环境。以下笔者先示范使用记事本来创建一个 Console 应用程序。

A-1

目的：示范使用记事本创建第一个 Console 应用程序。

(1) 打开记事本并输入以下的代码。这些程序主要的目的是使用 Writeln 与 Write 等函数在程序运行中显示信息，并通过 Readln 函数等待用户的输入信息以便结束程序。在“//”符号之后代表着单行的注释文本，编译器会忽略这个部分。

```
progran Project1;  //通过 prograr 标识符说明这是一个工程文件
{$APPTYPE CONSOLE} //将这个工程定义为 Console 应用程序
begin  //程序的开始
  Writeln('我的第一个 Console 应用程序');
  Write('请按下 Enter 键结束程序');
  Readln;  //等待用户输入按键，当输入 Enter 键则结束工程
end.  //程序的结束
```

(2) 将文字文件保存为“Project1.dpr”并通过 DCC32.EXE 编译程序进行编译。该程序在安装 Delphi 时会被安装于“Delphi\Bin”目录，如图 A.2 所示。

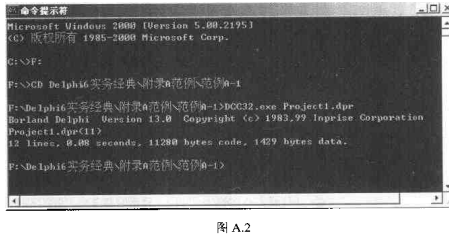


图 A.2

(3) 经过编译后，系统会自动产生一个与.DPR 相同文件名称的运行文件，您可以直接在命令提示字符下运行它。如果您要结束该程序只要按下【Enter】键即可，如图 A.3 所示。



图 A.3

注意

在此程序中，我们使用 Writeln 与 Write 函数来显示信息，并通过 Readln 函数来读取用户输入的信息。如果您查询联机帮助文件，您可能会发现这些函数被定义在 System 单元文件中，那我们为什么不需要在工程文件中引用它呢？这是因为编译程序在编译的过程中会自动地为每一个应用程序引用 System 单元文件，您不需要额外的引用。

范例 A-2

目的：示范使用 Delphi 创建第一个 Console 应用程序。

STEP 1 打开 Delphi 并关闭目前所有的工程。从“File/New/Other...”打开“New Items”窗口。

STEP 2 从“New Items”窗口中选择“Console Application”，如图 A.4 所示。

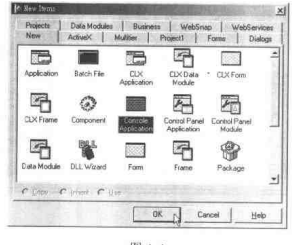


图 A.4

STEP 3 在工程文件中输入与范例 A-1 相同的代码。

STEP 4 保存并运行工程，此时您可得到与上一个范例相同的运行结果。



事实上，范例 A-1 与范例 A-2 的运行结果是完全相同的。如果要严格区分两者不同，应该是后者会自动产生.CFG 与.DOF 文件。.CFG 文件用来保存有关工程的 Configuration 信息，它是一个编译器可识别的设置文件，而.DOF 文件则代表各种编译器的设置信息。



A-3

目的：示范如何在 Console 应用程序传递命令行参数。

步骤 1 通过 Delphi 创建一个 Console 应用程序。

步骤 2 在工程文件中加入以下的代码。这段程序主要的功能是获取并显示 Console 应用程序自命令行所传递的参数。

```
program Project1;
{$APPTYPE CONSOLE}
uses SysUtils; //因为我们使用 IntToStr 函数
var
  i: Integer; //声明一个整数变量
begin
  Writeln('您总共输入了' + IntToStr(ParamCount) + '个参数');
  for i := 1 to ParamCount do
    Writeln('第' + IntToStr(i) + '个参数的内容为' + ParamStr(i));
  Readln; //等待按下 Enter 键结束程序
end.
```

步骤 3 保存并通过命令行运行工程。运行时程序可读取您所任意输入参数个数及其内容值，如图 A.5 所示。

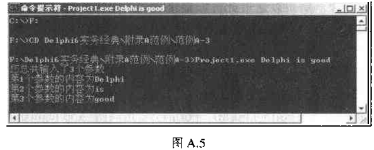


图 A.5



事实上，命令行的参数传递同样适用于 Windows 应用程序。您可以在程序的任何地方通过 ParamCount 函数与 ParamStr 函数获取命令行所传递的参数个数及参数内容。

A.3 Windows 应用程序

在此小节中笔者将为您介绍以 Delphi 的 VCL(Visual Component Library)所编写的

Windows 应用程序。事实上，本书其他的章节中已经花了很大的篇幅来介绍它。因此本小节仅是粗略的说明而已，关于每一个 VCL 组件的功能与使用方法请参考相关章节的详细说明。当您打开 Delphi 的 IDE 环境时，系统已经默认的帮您创建一个 Windows 应用程序，其包含一个工程文件、一个窗体文件、一个程序单元文件及其他的相关文件。在本小节中，我们所有讨论的只有工程文件与程序单元文件。您可以在 Delphi 的 IDE 环境中通过“Project/View Source”选项打开应用程序的工程文件，其内容如下所示。

```
program Project1; //通过 program 标识符说明这是一个工程文件

uses          //通过 use 标识符引用相关的程序单元文件
  Forms,      //与 TForm 对象有关的单元文件
  //用户自定义的程序单元文件，其附属于 Form1 窗体
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}  //说明与工程有关的资源文件

begin    //工程代码开始
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.    //工程代码结束
```

如果您将这些代码与 Console 应用程序的代码进行比较，您可能会发现这两种类型的应用程序事实上其工程文件并没有太大的差异。没错，这是因为工程文件是用来控制整个工程的运行流程，其真正的不同只是在于 begin 与 end 之间的程序块。在说明每一行语句前，我们先来看看什么是 Application。Application 是 TApplication 类的一个实例(Instance)，而 TApplication 类则被定义于 Forms 程序单元文件中。当 Delphi 工程被运行时，系统会自动创建一个 Application 对象，您可以通过此对象来获取与工程相关的信息，例如其 ExeName 属性就记录着应用程序的可执行文件名称。有关 Application 对象请参考相关章节的详细说明。我们现在已经知道什么是 Application 对象，接着再来看看上述工程中的代码就变得简单多了。

- “Application.Initialize;”是调用 Application 对象的 Initialize 操作方法，其主要目的是针对整个工程进行初始化的工作。
- “Application.CreateForm(TForm1, Form1);”是调用 Application 对象的 CreateForm 操作方法。此操作方法主要目的是创建一个 TForm1 类的实例，其名称为 Form1。这时候 Form1 对象还是被保存在内存并未被显示出来。如果工程中包含 n 个自动创建的 TForm 对象，那么 CreateForm 操作方法就会被调用 n 次，只是参数的内容不同而已。
- “Application.Run;”是调用 Application 对象的 Run 操作方法，其主要目的是开始运行工程。这个时候系统会显示主要的 TForm 对象，在这个范例中其主要的 TForm 对象为 Form1。

接下来，我们来看看 Unit1 程序单元文件的内容为何？

```
unit Unit1; //通过 unit 标识符说明这是一个程序单元文件

{通过 interface 标识符来区分程序单元文件的 Section，到 implementation 为止，此 Section
主要是用来声明常量、变量、过程、函数或定义一些类型，例如 TForm1 则是一个派生自 TForm 的类型。}
interface

uses //通过 use 标识符引用相关的程序单元文件
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs; //这些单元文件都是系统自动加入的
type //TForm1 是衍自 TForm 的类型，它可以再加上其他的 VCL 组件。
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var //变量声明
  Form1: TForm1; //声明一个 TForm1 类型的变量，名称为 Form1
{ interface 的 Section 结束}

implementation{ implementation 的 Section 开始}

{$R *.dfm} //这个编译指引连接 Unit1 的窗体文件

end. //单元文件结束，请特别注意，它是 "." 而不是 ";"。
```

因为这个程序单元文件 Unit1.pas 所对应的窗体文件中并没有加入其他的组件，所以这个程序单元文件的内容非常的简单。有关程序单元文件请参考下一个小节的详细说明。前面曾经说明工程文件是用来控制整个工程的运行流程，因此您可以在程序运行前通过工程文件做一些变量初始化、用户权限、应用程序试用期或是程序运行前显示标题窗口等。

范例 A-4

目的：示范如何通过工程文件来控制应用程序的试用期限。

STEP 1 创建一个新的工程。

STEP 2 从 "Project/View Source" 打开工程文件并且在工程文件中加入以下代码，这段代码主要的目的是获取系统时间并判断是否在试用期内。

```
{因为我们使用 DateTimeToString,Now,Showmessage 等函数，因此必须引用相关的程序单元文
件 SysUtils 与 Dialogs}
  SysUtils, Dialogs,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
var
  myDate : String; //声明一个字符串变量
begin
```

```
//获取今天的日期
DateTimeToString(myDate,'yyyymmdd',Now);
if myDate >'20010710' then Showmessage('程序已超过试用期限')
  else begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
  end;
end.
```

步骤 3 保存并运行工程。假如您计算机的系统时间超过程序中所设置的试用期，那么程序将不会被运行而且会显示如图 A.6 所示的对话框。



图 A.6 程序试用期限控制实例

范例 A-5

目的：示范如何通过工程文件来显示程序运行前的标题窗口。

步骤 1 创建一个新的工程。

步骤 2 从“File/New/Form”为此工程另外添加一个 TForm 对象，其相关属性设置如表 A.1 所示。

表 A.1

对象(Name 属性)	属性名称	属性设置值
TForm	Name	Form2
	BorderStyle	bsNone
	Height	214
	Width	343

步骤 3 从“Project/Options”打开“Project Options”窗口，并将 Form2 设为“Available Form”。

步骤 4 从“Project/View Source”打开工程文件，并且在工程文件中加入以下的代码，这段代码主要是手动创建 TForm2 对象并加以显示，最后当其他的 TForm 被加载内存后，再将 TForm2 对象释放，以达到程序运行前标题窗口的效果。

```
begin
  with TForm2.Create(Application) do //手动创建 TForm2 对象
    try
      Show;   //显示 TForm2
```

```
    Refresh;//更新TForm2
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
  finally
    Hide;  //隐藏TForm2
    Free;  //释放TForm2
    Application.Run;
  end;
end.
```

步骤 5 保存并运行工程。因为这个工程只有加载一个 TForm1 对象，因此“程序运行前的标题窗口”并不会停留太久。如果您无法清楚地看到它的效果，那您可以多加入一些 TForm 对象再试一下。

注意 范例 A-5 所示范的“标题窗口”是非常粗糙的，您可以使用其他的对象，如 TPanel 对象或 TImage 对象将其修饰的更美观一些。关于 TPanel 对象或 TImage 对象请参考相关章节的说明。

A.4 工程与程序单元文件

前面两个小节中，笔者介绍了 Console 应用程序与 Windows 应用程序。通过这两个小节的范例说明，相信您对工程文件(.DPR)应该有了相当程度的了解。因为您无法通过工程文件记录所有的代码，因此我们会将一个程序分割为多个程序单元文件。实务上，我们会按照窗体(TForm)来作区分，因为一个窗体必定对应于一个程序单元文件。当然，我们也可以将常用的函数或过程保存于同一个程序单元文件中。每一个程序单元文件被编译后会产生一个.DCU 文件(Delphi Compiled Unit)，所有的.DCU 文件经过连接就可以变成.EXE 或是.DLL 文件。这样的分割方式除了让程序变得更具可读性与支持多人共同开发外，最大的好处是代码不需要公开，因此其他程序设计师只要有.DCU 文件就可以进行连接的操作，而不需要程序单元文件的源代码。一般来说，一个程序单元文件除了程序单元文件头外，共分为 interface、implementation、initialization 和 finalization 等四个部分。其中 initialization 和 finalization 是可以省略的，以下就这几个部分加以说明：

- *interface*

程序单元的 interface 是从 interface 标识符到 implementation 标识符为止，此段主要是用来声明常量、变量、过程、函数或定义一些类型。其中过程与函数在 interface 中只需要声明即可，至于其内容的部分则写在 implementation 中。

- *implementation*

程序单元的 implementation 是从 implementation 标识符到 initialization 标识符为止，此段主要是用来定义在 interface 部分所声明的过程与函数。您可以省略过程或函数的参数行，否则必须与 interface 部分所定义的完全相同。

initialization

程序单元的 initialization 是从 initialization 标识符到 finalization 标识符为止，此区段主要是用来做变量的初始化工作。Initialization 部分可以包含多个语句，系统会按照其出现的顺序运行。如果一个工程中包含多个程序单元文件，而且每一个程序单元文件中都包含 Initialization 部分的话，那么系统会按照程序单元文件的顺序依次做初始化的操作。如果应用程序初始化时发生错误时 Initialization 部分的代码并不一定保证被运行。

finalization

程序单元的 finalization 是从 finalization 标识符到 end.为止，此段主要是用来释放 Initialization 部分所配置的系统资源。如果一个工程中包含多个程序单元文件，而且每一个程序单元文件中都包含 finalization 部分的话，那么系统会按照程序单元文件的顺序反向操作。在应用程序结束时 finalization 部分保证一定会被运行。

工程文件与程序单元文件或程序单元文件之间可以通过 uses 标识符相互进行引用。程序单元文件的引用可以在 interface 部分或 implementation 部分，程序单元文件的引用必须注意下列各点：

- uses 标识符之后的程序单元名称必须以逗号(,)加以分开，并以分号(;)作为结束。
- 在每一个程序单元名称的后面可以通过 in 标识符说明实际的文件名称，您可以使用绝对路径或相对路径来描述此文件名称。
- 如果两个程序单元文件中同时定义相同名称的变量、过程或函数时。访问时您必须加上该程序单元名称以作为标识，例如 Unit1.myAge 或 Unit2.myAge 等。
- 两个程序单元文件可以相互引用，但该程序单元文件名称不可以在 interface 部分形成循环。为避免发生这种循环参考，建议您将其中一个引用写在 implementation 部分。

A.5 程序注释

在前面四个小节中笔者以宏观的角度介绍了 Object Pascal 与 Delphi 之间的关系，从本小节开始笔者要把重点放在 Object Pascal 的程序编写技巧上。在正式编写程序之前笔者希望您能注意以下的两件事：

清晰的程序编写风格

Delphi 6 是使用 Object Pascal 的语法。因此可以允许将一行语句分成很多行来编写。虽然这样的写法编译程序同样可以编译，但别人看了可能会很辛苦的。请比较以下两种写法：

```
procedure
TForm1.          FormCreate
(Sender: TObject);begin Color:=clYellow;end;
(不建议您使用的写法)

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    Color:=clYellow;
end;
```
（建议您使用的写法）

适度的程序注释

除了养成良好的编写风格外，适度的注释也是必要的。Delphi 6 提供三种方法作为注释之用：

- // ：是作为单行的注释，在//之后的文字，都视为注释。
- {}：是作为块的注释。在 { 与 } 之间的文字都视为注释。
- (* *)：是作为块的注释，在(* 与*)之间的文字都视为注释。

注意 如果在注释符号{或(*之后紧接着$符号时，那表示它是一个编译指引，也就是说 Delphi 的编译器并不会将其视为一般的代码。例如{$R *.res}或{$APPTYPE CONSOLE}等，前者说明与工程有关的资源文件，而后者则是将工程定义为 Console 类型的应用程序。

A.6 变量声明

程序运行过程中会更改其内容者我们称之为变量，其与常量有显著的不同。要声明一个变量时要特别注意下列事项：

- 变量名称可以是任意长度，但只有前 255 个字符是有效的。如果超过 255 个字符，编译器会忽略第 255 个以后的字符，但并不会发生错误。
- 变量名称可以是由英文字母、数字及下划线(_)所组成的，但变量名称前缀必须以英文字母、下划线(_)为开头。
- 在 Delphi 中的变量名称并无字母大小写之分(Case-insensitive)。
- 不可使用保留字(Reserved words)、指令(Directives)或 Object Pascal 已经定义的对象名称作为变量名称（见表 A.2 和表 A.3）。

表 A.2

保留字 (Reserved words)		
and	array	as
asm	begin	case
class	const	constructor
destructor	dispinterface	div
do	downto	else
end	except	exports
file	finalization	finally
for	function	goto

续上表

保留字 (Reserved words)		
if	implementation	in
inherited	initialization	inline
interface	is	label
library	mod	nil
not	object	of
or	out	packed
procedure	program	raise
record	repeat	resourcestring
set	shl	shr
string	then	threadvar
to	try	type
unit	until	uses
var	while	with
xor		

表 A.3

指令 (Directives)		
absolute	abstract	assembler
automated	cdecl	contains
default	dispid	dynamic
Export	external	far
forward	implements	index
message	name	near
nodefault	overload	override
package	pascal	private
protected	public	published
read	readonly	register
reintroduce	requires	resident
safecall	stdcall	stroed
virtual	write	writeonly

- 变量声明除了变量的命名规则外，尚有一个重点，那就是设置变量类型，至于 Delphi 所提供的数据类型，请参考下一章的详细介绍与说明。设置变量的类型必须注意下列三个重点：
 - 设置变量的类型必须在 var 保留字之后。在 var 之后，可以同时声明多个变量。
 - 先设置变量名称再设置变量类型，变量名称与变量类型之间用冒号(:)加以分开，在语句最后以分号(;)作为结束符号。
 - 同一类型的变量可一起声明，但变量与变量之间必须用逗号(,)加以分隔。

```
//在 var 之后可声明多个变量
var
  //同类的数据类型可一起声明给设置初值
  I,J,K,Temp:=0:Integer;
  MyName:String;
  MySex:Boolean;
```

A.7 常量声明

就如同前面章节所说的，变量是一个在程序运行期间可以改变的数。相对的，常量就是一个在程序运行期间不会改变的数。例如 π=3.1416...，这个数在程序运行时期是不易被改变的，因此就可以将它设为常量。常量的命名规则与变量相同，要声明一个常量同样也有下列几点需要特别注意的：

- 设置常量的类型必须在 const 保留字之后。在 const 之后，可以同时声明多个常量。
- 先设置常量名称再设置常量值，常量名称与常量值之间用等号(=)加以分开，在语句最后以分号(;)作为结束符号。
- 一般来说，常量一次仅声明一个，如果两个常量的内容是相同的，必须经过两段的声明。一般来说，常量是不需要特别指定其数据类型，系统会根据设置值自动设置适合的数据类型。

```
const
  PI = 3.1416;                    //设为实数类型(Real)
  MyName = '宋次郎';              //设为字符串类型(String)
  MyNickName = MyName;            //设为字符串类型(String)
  MyAge = 18;                     //设为整数类型(Integer)
```

A.8 语句(Statement)

通常一个程序是由多个语句所组成，而每一个语句就代表着程序的一个算法操作。Delphi 的语句种类可分为简单语句与结构化语句，分述如下：

简单语句

简单语句顾名思义就是 Delphi 中最简单的语句，它不能再包含其他的语句。简单语句包括指定叙述、子程序调用及 goto 跳转指令。

结构化语句

结构化语句与简单语句最大的不同在于前者可以由其他的语句所构造而成。它可以是以循序、条件判断或重复运行等方式出现。

A.8.1 赋值语句

在程序设计时常常会需要指定某个变量值，在 Delphi 中的赋值语句是“:=”，在每一个赋值语句后面要用“;”来做为结束符号。在使用赋值语句时需注意以下几点：

1. 左边与右边的数据类型必须一致。
2. 左边不可为常量。
3. 右边可以为常量、变量、表达式、对象属性等。

A.8.2 过程与函数调用语句

除了赋值语句外，Delphi 所提供的另一个简单语句，就是过程或函数的调用语句。在说明过程与函数的调用语句之前，我们先来看看什么是过程(procedure)与函数(function)。过程是由一个或多个语句所组合而成，通常是一个特定的计算或工作。函数的功能与过程类似，其最大的不同在于过程没有返回值，而函数必定会有返回值。有关程序与函数请参考章节 A.9 的详细说明，这里仅示范如何调用过程与函数。

```
//调用一个名称为 myProcedure 的过程，其不包含参数行
myProcedure;
//调用一个名称为 myProcedure2 的过程，其包含参数行
myProcedure2(123,'abc');
//调用一个名称为 myFunction 的函数，其不包含参数行
myResult := myFunction;
//调用一个名称为 myFunction2 的函数，其包含参数行
myResult := myFunction2(123);
//调用一个名称为 myFunction2 的函数，其包含参数行，但舍弃返回值。
myFunction2(123);
```

A.8.3 Goto 语句

Goto 语句可以强迫程序跳转到指定的标签位置。笔者并不建议您使用 goto 语句，最主要原因是因为它会破坏整个程序的结构化。下面的程序片段示范如何使用 goto 语句。使用 goto 语句之前，您必须先使用 Label 关键字定义标签以利程序跳转。

```
Label myLabel1; //定义标签
.....; //可包含其他的程序语句
goto myLabel1; //强制跳转到 myLabel1 标签。
```

A.8.4 复合语句

所谓复合语句是指由多个语句所形成的块，在此块中的语句会按照其出现的先后顺序来

运行。在 Delphi 中只要是使用 begin 和 end 所包含的块都称为复合语句。在很多情况都可以使用它，以提高程序的可读性。其中，with 语句是最常被使用的复合语句，它提供一种非常简便的程序编写方式。

```
//通过 TDBGrid 对象找到其所对应的 TDataSet 并进行添加记录
DBGrid1.DataSource.DataSet.Append;
DBGrid1.DataSource.DataSet.FieldByName('ID').AsString := 'A001';
DBGrid1.DataSource.DataSet.FieldByName('Name').AsString := 'Carlos';
DBGrid1.DataSource.DataSet.Post;
//使用 with 语句的简化结果
with DBGrid1.DataSource.DataSet do begin
     Append;
     FieldByName('ID').AsString := 'A001';
     FieldByName('Name').AsString := 'Carlos';
     Post;
end;
```

with 语句也可以同时使用于多个对象，假如其中的语句可以同时适用于多个对象时，该语句还是只会被运行一次，而且是反应于最后一个对象。

```
with Edit1, Edit2, Edit3 do begin
    Text := 'Carlos';
    Color := clYellow;
end;
```

以上面的范例而言，其中的两个语句都会反应于 Edit3 对象。您可能会觉得很奇怪，为什么是最后一个对象呢？其实正确的说法应该为“最近”的一个对象，因为上面的范例可以改写为下面范例的形式，换言之，这两种写法对 Delphi 而言是相同的。

```
with Edit1 do
  with Edit2 do
    with Edit3 do begin
        Text := 'Carlos';
        Color := clYellow;
    end;
```

A.8.5 条件语句

Delphi 所提供的条件语句包括 if 及 case。其中 if 语句是针对单一判断条件，而 case 语句则是针对多个判断条件。您可以想象 case 语句是由多个 if 语句所组合而成的。分述如下：

- If 判断语句可以依判断结果分别运行各个不同的代码。如果语句只有一条则不需要 begin ... end，其中 else 可以根据需要而决定是否套用。

```
If 判断式 then
   begin
       语句1;
       ……
```

```
        语句n;
    end
else
    begin
        语句1;
        ……
        语句n;
    end
```

下面范例主要是通过简单判断的程序来说明 if …else …end 语句的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
var
    MySex:Char;
begin
   MySex:='M';
   if MySex = 'M' then
     ShowMessage('林先生，你好!!')
   else if MySex = 'F' then
     ShowMessage('林小姐，你好!!')
   Else
     showMessage('where do you come fron ?');
end;
```

- Case 判断语句，简单地说，它是多个 if 语句的简化版本。它可以按判断条件分别运行各种不同的代码。如果语句只有一条，则不需要使用 begin 和 end。Case 判断语句适用的数据类型仅限于 Integer type、Char type、enumerated type 及 subrange type 等数据类型。其语法如下：

```
case 判断式 of
   限制式1:语句1;
   限制式2: begin
                    语句2;
                    语句3;
                end;
    else   //当以上的限制式都不成立时运行
      : 语句4;
end;
```

下面范例主要通过简单判断的程序来说明 case 语句的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
var
    MySex:char;
begin
   MySex:='M';
   case MySex of
```

```
    'M': ShowMessage('林先生，你好！');
    'F':ShowMessage('林小姐，你好！');
  else
    showMessage('where do you come from ?');
  end;
end;
```

A.8.6 循环语句

循环可以重复运行一连串的语句一直到达循环的结束为止，因此使用循环语句最重要的一件事就是控制其结束条件。Delphi 所提供的循环控制包括 Repeat 循环语句、While 循环语句及 For 循环语句等。

A.8.6.1 Repeat 循环语句

Repeat 循环可重复运行某些特定的代码直到判断条件为 True 才结束，通常是用于至少运行一次的循环。其语法如下：

```
Repeat
   语句 1;
   语句 2;
   ……
   语句 n;
Until 结束条件;
```

下面范例主要通过 9*9 乘法表的编写来说明 Repeat.....Until 循环语句的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
var
   I,J :Integer;   //设置两个循环变量
begin
  I:=0;            //设置循环变量初值
  Repeat
    I:=I+1;        //循环变量递增，否则循环永远不会结束
    J:=0;          //设置循环变量初值
    Repeat
      J:=J+1;      //循环变量递增
      Canvas.TextOut(I*50,J*20+40,IntToStr(I)+'*'+IntToStr(J)+'='+
                                  IntToStr(I*J));
    Until J = 9;   //循环变量递增
  Until I = 9;     //循环变量递增
End;
```

A.8.6.2 While 循环语句

While 循环可重复运行某些特定的代码，每次进入 While 循环都会进行条件判断，只有判断条件为 True 才进入循环。通常用于运行次数不确定的循环，其语法如下：

```
While 判断条件=True do
    begin
        语句 1;
        语句 2;
        ……;
        语句 n;
    end;
```

本范例主要通过 9*9 乘法表的编写，来说明 While …do 循环语句的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
var
   I,J :Integer;              //设置两个循环变量
Begin
  I:=0;                       //设置循环变量初值
  while I < 9 do              //判断结束条件
    begin
      I:=I+1;                 //循环变量递增，否则循环永远不会结束
      J:=0;                   //设置循环变量初值
      while J < 9 do          //判断结束条件
        begin
          J:=J+1;             //循环变量递增
          Canvas.TextOut(I*50,J*20+40,IntToStr(I)+'*'+IntToStr(J)+'='+
                                      IntToStr(I*J));
        end;
    end;
end;
```

A.8.6.3 For 循环语句

For 循环可重复运行某些特定的代码，它可以指定循环计数器的上、下限及重复次数，因此它通常用于运行次数确定的循环，其语法如下：

```
For I:=计数器初值 to 计数器结束值 do
    begin
        语句 1;
        语句 2;
        ……;
        语句 n;
    end;
```

如果你的计数器是由大到小，那你必须用下面的语法。

```
For I:=计数器初值 downto 计数器结束值 do
    begin
        语句 1;
        语句 2;
        ……;
```

```
        语句n;
    end;
```

本范例主要通过9*9乘法表的编写来说明For循环语句的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
var
   I,J :Integer;          //设置两个循环变量
begin
  For I:=1 to 9 do        //设置循环变量初值及结束值
  For J:=1 to 9 do        //设置循环变量初值及结束值
     Canvas.TextOut(I*50,J*20+40,IntToStr(I)+'*'+IntToStr(J)+'='+
                                 IntToStr(I*J));
end;
```

A.8.6.4 循环中止语句

除了正常的结束条件外，您也可以使用Break、Continue或是Exit等语句来控制循环，其中Break语句主要是用于跳出循环、Continue语句是用于省略某个变量计数器、Exit语句不但会结束循环而且会结束目前所在的过程、函数或工程。下面的程序段将说明如何使用Break、Continue或Exit等语句来控制或中止循环的进行。

```
//Break范例说明，本程序片段只会印出1,2及'结束'。
    For I:=1 to 5 do
      Begin
        if i = 3 then Break;
        ShowMessage(IntToStr(I));
      end;
    ShowMessage('结束');
//Continue范例说明，本程序片段只会印出1,2,4,5及'结束'。
    For I:=1 to 5 do
      Begin
        if i = 3 then Break;
        ShowMessage(IntToStr(I));
      end;
    ShowMessage('结束');
//exit范例说明，本程序片段只会印出1,2。
    For I:=1 to 5 do
      Begin
        if i = 3 then exit;
        ShowMessage(IntToStr(I));
      end;
    ShowMessage('结束');
```

A.8.7 异常处理语句

除了上述所介绍的语句外，本小节笔者将特别为您介绍一些程序编写过程中非常重要的特殊语句，那就是 Object Pascal 所提供的异常处理机制，这样的机制能让您的程序轻易的应付各种突发状况，例如：不合法的除法运算、打开一个不存在的文件或是将数据写入一个已满的磁盘等。当 Object Pascal 程序发生这种特殊的运行时期错误时，程序就会触发一个异常。换言之，程序在此时创建了一个异常对象。一般来说，当一个异常被触发时，您可以选择“清理异常”、“处理异常”或者两者同时使用的处理方式。

- 当程序发生异常时，“清理异常”的处理方式主要的目的是确保运行环境能处于一个稳定的状态。这样的处理方式通常是释放相关的系统资源，例如释放先前所配置的内存或关闭先前所打开的文件等。
- 当程序发生异常时，“处理异常”的处理方式是针对特定的异常类型，而采取相对应的处理方式。这种处理方式的目的是清除错误并消灭系统所产生的异常对象，以便让程序能继续顺利的运行。例如：当程序运行某个除法运算时产生一个分母为 0 的异常时，您可能消灭该异常对象并将运算结果设为 0。

Object Pascal 所提供的异常处理语句为“try…except…end; ”除了这种异常处理语句外，Object Pascal 另外提供一种资源保护语句。资源保护语句是 Object Pascal 为了能确保程序在发生异常的情况之下，仍然可以释放先前所配置的资源而设立的一种异常处理语句，资源保护语句为“try…finally…end; ”。您只要将需要被保护的代码写在 finally 与 end 之间，那么程序保证一定会运行它，无论发生异常与否。

A.8.7.1 try…except…end 语句

为什么需要异常处理机制，难道传统的程序编写方式无法处理吗？以先前所讨论的“不合法的除法运算”为例，如果您不使用 Object Pascal 所提供的异常处理机制，那么您必须在运行除法运算之前先行判断分母是否为 0。若分母为 0 则做出适当的处理工作；反之，才能运行真正的运算工作。也许面对这种简单且确定的问题，您可以通过如此传统的方式来处理。请想一想，您如何使用这样传统的方式处理“将数据写入一个已满的磁盘”的问题呢？在您想到答案之前，我们先来讨论 Object Pascal 所提供的异常处理语句。try…except…end 异常处理语句的语法如下：

```
try
    语句 1;
    语句 2;
    ……
    语句 n;
except
    on  异常类 1  do 语句 a
    on  异常类 2  do 语句 b
    //你可以在这里进行异常处理. 如果你不处理，那就表示你要忽
    //略这个异常
end;
```

从 try 关键字开始正式进入异常保护块，程序会监控所有置于 try 关键字和 except 关键字之间的代码。当这个代码发生运行时期错误时，系统则创建一个异常对象并将程序的控制权交给 except 与 end 之间的代码。您可以将异常处理的代码写在这个块中，当然，您可以忽略异常类而给予相同的处理方式，更可以针对不同的异常类给予不同的处理方式。当程控权跳离 except 关键字与 end 关键字之间的块时，也就表示该异常对象已被释放。假如您想忽略任何异常时，您并不需要在 except 关键字与 end 关键字间填入任何的代码。以下将以实例来说明如何使用 try...except...end 异常处理语句来处理一个日期类型的转换错误。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  try
    //数据类型错误，会发生一个转换性的错误
    MyDate := StrToDateTime('xx/xx/1999');
  except
    //根据异常的类型来给予不同的处理
    on MyError: EConvertError do
      ShowMessage('异常的 ClassName 是'+MyError.ClassName+ #10 + '异常的
Message 是'+MyError.Message);
  end;
end;
```

当您的程序在 Delphi 的集成性开发窗口下运行，如果您不想程序因为发生异常而被中断的话，如图 A.7 所示。那请你选择菜单“Tools/Debugging Options..”，打开“DeBugging Options”对话框，并将“Language Exceptions”页标签中的“Stop on Delphi Exceptions”取消即可，如图 A.8 所示。



图 A.7 程序因异常而中断

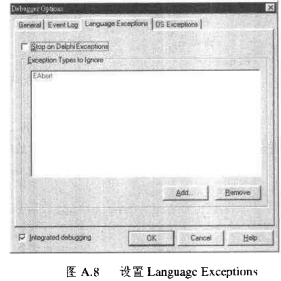


图 A.8 设置 Language Exceptions



Object Pascal 定义了许多异常类，在这里无法一一的详述，如果您有兴趣可以查阅帮助说明。大体上来说，只要是以英文字 E 开头，以 Error 结束的，大部分都是 Object Pascal 所提供的异常类，如 EdateTimeError、EdataBaseError、EconvertError、EdivByZero、ErangeError 及 EintOverFlow 等都是。

Object Pascal 允许您在某个 try...except...end 异常处理语句中再包含另外一个异常处理语句，如下所示：

```
try {第一层的保护块起点}
     语句 1;
     try {第一层的保护块起点}
          语句 2;
     except{第二层的异常处理程序}

     end;
      ……
     语句 n;
except {第一层的异常处理程序}
     //不分异常类直接显示异常信息。
     on E:Exception do Showmessage(E.Message);
end;
```

A.8.7.2 try...finally...end 语句

Try...finally...end 语句也是异常处理的一种，它通常用于运行一些必定要被运行的代码以确保系统资源的正确运用。例如您为了确保程序中动态创建一个对象或动态配置的内存，能在程序块结束前加以释放。那么这个时候您就可以使用 Try...finally...end 语句。Try...finally 语句的语法如下：

```
try
     语句 1;
     语句 n;
finally
     //你可以在这里设计一些必要的代码
end;
```

下面程序段示范如何通过 try...finally...end 语句来显示一个动态所创建的 TForm 对象，并保证该 TForm 对象在关闭后一定能释放被占用的相关系统资源。

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
   with TForm2.Create(Application) do   //动态创建一个 TForm 对象
   try
       ShowModal;   //Show 出这个 Form
   finally
      Free;        //释放这个 Form
   end;
```

```
end;
```

Try…finally…end 语句同样能与 try…except…end 异常处理语句合并使用，如下所示：

```
try
    try
        语句1;
        语句n;
    except
        on E: Exception do  //加入异常处理的代码
    end;
finally
    //释放相关的系统数据
end;
```

A.8.7.3 自定异常类

Object Pascal 除了允许您自行定义异常类，最简单的类定义方式就是继承 SysUtils 中所定义的 Exception 类。除此之外，您也可以在程序的任何地方自行触发任意的异常类。以下范例将示范如何触发一个用户自行定义的异常类。

```
(略)
type
  //自行定义一个异常
  ECarlos = Class(Exception);
(略)
procedure TForm1.Button1Click(Sender: TObject);
var
  A, B : Integer;
  Str : String;
begin
  A := 0;
  B := 3;
  try
    if A < B then
      raise ECarlos.Create('第一个参数不能小于第二个参数');
  except
    on E : Exception do begin
      Str := '异常的信息=' + E.Message - #10;
      Str := Str + '异常的类=' + E.ClassName;
      Showmessage(Str);
    end;
  end;
end;
```

A.8.7.4 再次触发异常

Object Pascal 允许您通过 raise 保留字再次触发目前所产生的异常。以下范例将示范如何再次触发一个异常类。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A, B : Integer;
  F : Extended;
begin
  try
    try
      A := 3;
      B := 0;
      F := A / B; //应该会产生一个 EZeroDivide 异常
      Showmessage(FloatToStr(F));
    except
      on E: Exception do begin
        Showmessage(E.ClassName);
        raise; //不做任何处理，再次触发异常
      end;
    end;
  except
    on E : Exception do Showmessage(E.ClassName);
  end;
end;
```

A.9. 过程(Procedure)与函数(Function)

在写程序的过程中，常常需要重复编写相同或类似的程序段。为了节省系统开发的不必要资源浪费与日后维护的考虑，我们通常会将这些代码依其用途分别编写成一个过程(Procedure)或是一个函数(Function)来使用。这两者之间有何不同呢？简单地说，过程就像妈妈在出门前请你去做某件事，可能是拿水果去给隔壁的阿姨，因此您只需要运行某件事即可，不需要对妈妈做响应(因为妈妈不在家)。相对的，函数就好像是妈妈给你钱，请你去买酱油。因此当你买完酱油后一定要将它拿给妈妈。这就是过程(Procedure)和函数(Function)之间最大的不同之处。

过程(Procedure)和函数(Function)统称为子程序，它们都是一个完整的程序块，因此它们与其他的变量或常量一样可以被调用或使用于程序中的任何地方。因为函数有返回值，因此您可以直接调用某函数，并将其返回值指定给其他变量或函数。

```
//调用一个有整数返回值的函数并将其返回值指定成整数变量 i。
i := GetMyAge;

//调用一个有字符串返回值的函数并将其返回值指定给其他函数。
```

```
Str := Copy(GetMyName, 1, 5);
```

上面两个范例示范如何调用函数并将其返回值指定成变量或函数。以第二个例子而言，Str 变量只会获取 GetMyName 函数返回值的前面 5 个字而已。如果您舍弃函数的返回值，那么过程与函数的调用方法则是一样的。

```
//调用一个有整数返回值的函数，但舍弃函数的返回值
GetMyAge;

//调用一个指定名称的过程。
DoSomeJob;
```

A.9.1 过程(Procedure)的定义与声明

过程(Procedure)与变量或常量在使用上有些稍微的不同，通常使用过程必须经过定义与声明两个操作。前者只定义一个函数原型(Prototype)，您也可以说是声明一个过程头(Procedure Header)。而后者才是真正的编写代码，您也可以称它为子程序的程序主体(Body)或是程序块。笔者在此使用定义与声明这两个名词来说明它。一般来说，在定义过程时您必须说明该过程的名称、其所需要的参数个数及每一个参数的数据类型。在声明过程的主体时，您同样需要说明过程名称或参数行。每一个过程主体可包含其 Local 的变量声明，如下所示：

```
Procedure 过程名称(参数行);
Var  //假如没有 Local 的变量需要声明，那么此行可以省略。
   //Local 的变量声明;
Begin
   //加入过程的处理代码
End;
```

范例 A-6

目的：通过一个比较参数值大小的范例，说明如何创建与调用过程。

做法 1 创建一个新的工程。

做法 2 为 TForm 对象创建一个 MyCompare1 过程，该过程包含两个整数类型的参数。

```
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    //声明过程的头
    procedure MyCompare1(First, Second : Integer);
  end;
(略)
//定义过程的主体
```

```
procedure TForm1.MyCompare1(First, Second : Integer);
begin
  if First > Second then Showmessage('第一个参数比较大')
  else  Showmessage('第一个参数小于或等于第二个参数');
end;
```



因为 MyCompare1 过程是属性 TForm1 对象的一部分，所以过程的标头必须放在 TForm1 对象的类型定义中，而且过程的主体也必须加上 TForm1.的限定词。

STEP 3 为 Unit1 单元文件创建一个 MyCompare2 过程，该过程的主体与 MyCompare1 过程完全相同。

```
var
  Form1: TForm1;
  procedure MyCompare2(First, Second : Integer);
(略)
procedure MyCompare2(First, Second : Integer);
begin
(略)
end;
```



因为 MyCompare2 过程是属性 Unit1 的一部分，所以过程的头不需要加在 TForm1 对象的类型定义中，而且过程的主体也不需加上 TForm1.的限定词。

STEP 4 为 TForm 对象加入一个 TButton 对象并在该对象的 onClick 事件处理过程调用 MyCompare1 与 MyCompare2 两个过程。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //调用 MyCompare1
  MyCompare1(1,2);
  //调用 MyCompare2
  MyCompare2(1,2);
end;
```


对于 Button1 对象的事件处理过程而言，调用 MyCompare1 与 MyCompare2 两个过程的方式都是一样的，那么您可能会问？既然是一样，那到底应该使用哪一个方法才正确呢？其实，这并没有绝对的标准答案。如果您的工程中只有一个程序单元文件而且程序的内容也很简单，那么您可以使用 MyCompare1 过程的创建方式。如果您的工程很大而且也有很多用户自定义的过程或函数时，那么笔者会建议您将所有的子程序集中或分类于一个或多个程序单元文件中以利管理。除此之外，其他单元文件的调用方式也是一个考虑因素。

STEP 5 通过“File/New/Form”为此工程加入另一个 TForm 对象，其默认名称为“Form2”。

STEP 6 为 Form2 对象加入一个 TButton 对象并且在该对象的 onClick 事件处理过程中调用 MyCompare1 与 MyCompare2 两个过程。

```
procedure TForm2.Button1Click(Sender: TObject);
```

```
begin
  //调用 MyCompare1
  Form1.MyCompare1(1,2);
  //调用 MyCompare2
  MyCompare2(1,2);
end;
```



请特别注意，如果我们通过其他的程序单元调用 MyCompare1 与 MyCompare2 两个过程时，其调用方式是不一样的。前者必须加上 Form1 的对象名称，而后者却不需要。

A.9.2 函数(Function)的定义与声明

函数(Function)与过程一样都必须经过定义与声明两个操作。在定义函数时同样需要说明该函数的名称、其所需要的参数个数及每一个参数的资料类型。最重要的一点，您一定要说明其返回值的数据类型。在声明函数的本体时，您同样需要说明函数的名称或参数行。每一个函数主体同样可以包含 Local 的变量声明，而且您一定要指明该函数的返回值语句否则编译时会产生警告性的错误信息。

```
Function  函数名称(参数行):返回值类型;
Var  //假如没有 Local 的变量需要声明，那么此行可以省略。
   //Local 的变量声明;
Begin
   //加入过程的处理代码
   Result := 返回值;
End;
```



您除了可以使用 Result 来设置函数返回值外，您也可以直接使用函数名称来取代 Result 以设置该函数的返回值。当您设置为某个函数的返回值时，程序的控制权并不会马上交还，而是会继续往下运行，一直到程序块结束为止。例如下面函数的返回值应该为 2，而不是 1。

```
function  TForm1.MyFunction:Integer;
begin
  MyFunction:= 1;
  ShowMessage('您猜这个对话框会不会显示');
  MyFunction:= 2;
end;
```


A-7

目的：通过一个比较参数值大小的范例，说明如何创建与调用函数。在范例 A-6 所强调的过程特性同样适用于函数，同样的用法笔者不再赘述，请您另外参考范例 A-6 的详细说明。

STEP 1 创建一个新的工程。

STEP 2 为 TForm 对象创建一个 MyCompare1 函数，该过程包含两个整数类型的参数及一个布尔类型的返回值。

```
var
  Form1: TForm1;
//声明过程的头
function MyCompare1(First, Second : Integer): Boolean;

implementation
{$R *.dfm}
//定义过程的主体
function MyCompare1(First, Second : Integer): Boolean;
begin
  //比较第一个参数是否大于第二个参数
  Result := First > Second;
end;
```

STEP 3 为 TForm 对象加入一个 TButton 对象并在该对象的 onClick 事件处理过程调用 MyCompare1 函数。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MyCompare1(1,2) then
    Showmessage('第一个参数大于第二个参数')
  else
    Showmessage('第一个参数小于或等于第二个参数');
end;
```

A.9.3 参数的传递

前面已经详细的说明有关过程及函数的声明与使用方法，接着笔者将说明有关过程及函数的参数传递种类。在 Object Pascal 中参数传递可分为传值参数、变量参数及常量参数等三种。

A.9.3.1 传值参数

传值参数又称 Call By Value，这种传值参数会将外部所传来的值加以复制一份到此参数中。所以当您对此参数做任何改变，都不会影响外部所传来的最初参数值。

范例 A-8

```
//被调用的函数
Procedure Test(X,Y:Integer);
Begin
```

```
    X:=X+5;
    Y:=Y+10;
    ShowMessage('X=' + IntToStr(X) + #10 + 'Y=' + IntToStr(Y));
   //显示的结果 X=10,Y=20
End;
//函数的调用
procedure TForm1.Button1Click(Sender: TObject);
var X,Y:Integer;
begin
   X:=5;
   Y:=10;
   Test(X,Y);
   ShowMessage('X=' + IntToStr(X) + #10 + 'Y=' + IntToStr(Y));
   //印出的结果 X=5,Y=10
end;
```



如果被调用的函数或过程在调用者之后，那么你需要对被调用者先进行前缀声明(Forward Declarations)，函数的主题则不需重复声明变量部分。

A.9.3.2 变量参数

变量参数又称 Call By Address，这种变量参数会将外部参数的内存位置传入子程序。换句话说，它们是参考到同一个内存地址。所以当您对此参数做任何改变，其结果都会影响到外部所传递的最初参数值。变量参数与传值参数最大的不同在于被调用者的参数名称前必须加上 var 的保留字，而者调用时不以传递“值”，而要传递“变量名称”。以下的范例修改自范例 A-8，因此 Button1 对象的 onClick 事件处理过程的代码并没有改变。



A-9

```
//被调用的函数
Procedure Test(var X,Y:Integer);
Begin
    X:=X+5;
    Y:=Y+10;
    ShowMessage('X='+IntToStr(X)+#10#13+'Y='+IntToStr(Y));
   //输出的结果 X=10,Y=20，也影响到原来的值
End;
//函数的调用
procedure TForm1.Button1Click(Sender: TObject);
(略)
  ShowMessage('X=' + IntToStr(X) + #10 + 'Y=' + IntToStr(Y));
  //显示的结果 X=10,Y=20
```

```
end;
```

9.9.3.3 常量参数

常量参数又称 Constant Parameter，这种常量参数是一种只读的参数。换言之，您只能使用它而无法修改它的内容，常量参数与传值参数最大的不同在于被调用者的参数名称前必须加 const 的保留字。

```
//被调用的函数
Procedure Test(const X,Y:Tnteger);
Begin
    X:=X+5;    //您不可以修改参数的值，否则会出错
    Y:=Y+10;  //您不可以修改参数的值，否则会出错
End;
```

9.9.3.4 输出参数

输出参数与变量参数一样，也可以称为 Call By Address，这种输出参数会将外部参数的内存地址传入子程序。当被调用的子程序包含输出参数时，我们通常会舍弃该参数的初值。因为输出参数只负责告诉子程序数据输出的地址而已，并不负责数据的输入工作。输出参数与变量参数最大的不同在于被调用者的参数名称前必须加 out 的保留字，而非 var 保留字。

```
//声明一个包含输出参数的过程
procedure myOut(InStr : String; out OutStr : String);

implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  InputStr, OutStr : String;
begin
  InputStr := 'INPUT';
  OutStr := 'OUTPUT';
  //调用一个包含输出参数的过程
  MyOut(InputStr, OutStr);
  //OutStr=INPUT:This is a testing
  Showmessage('OutStr=' + OutStr);
end;
//自定义一个包含输出参数的过程
procedure myOut(InStr : String; out OutStr : String);
begin
    //空字符串，因为该函数会舍弃“输出参数”的初值
    Showmessage('OutStr=' + OutStr);
    OutStr := InStr + ':This is a testing';
```

```
end;
```

A.9.3.5 参数默认值

在子程序的头部分声明，您可以同时指定参数的默认值，但参数的默认值只适用于常量参数与传值参数。您只要在参数的声明后面利用等号指定其默认值即可。下列范例说明如何声明与调用一个包含参数默认值的子程序。

```
//自定义包含默认值的子程序
function MyDefault(X : Integer = 0; Y: Integer = 0):String;

implementation
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  //一般的调用方式
  Showmessage(MyDefault(10,20)); //X=10, Y=20
  //省略后面的参数
  Showmessage(MyDefault(10)); //X=10, Y=0
  //省略所有的参数
  Showmessage(MyDefault); //X=0, Y=0
end;

//自定义包含默认值的子程序
function MyDefault(X : Integer = 0; Y: Integer = 0):String;
begin
  Result := 'X=' + IntToStr(X) + ', Y=' + IntToStr(Y);
end;
```

注意 如果某子程序包含参数的默认值时，在声明多个具有相同数据类型的参数时，不能只使用逗号(,)加以隔开而已。而且包含默认值的参数必须置于所有参数的后面。

```
//合法的声明方式(包含默认值)
function MyDefault(X : Integer = 0; Y: Integer = 0):String;
//不合法的声明方式(包含默认值)
function MyDefault(X, Y: Integer = 0):String;
//合法的声明方式(不包含默认值)
function MyDefault(X, Y: Integer):String;

//不合法的声明方式(有默认值的参数必须置于所有参数的后面)
function MyDefault(X : Integer=0; Y:Integer):String;
//合法的声明方式
function MyDefault(X : Integer; Y:Integer=0):String;
```

A.9.3.6 数组参数

数组参数允许子程序接受大小不同的数组参数，数组参数可支持各种不同数据类型的变

量。而且数组参数同样支持变量参数的用法，换言之，您只要在参数的声明前面加上 var 保留字，则此数组参数即为“变量参数”。下列范例说明如何声明与调用数组参数的子程序。

```
//自定义一个数组参数的子程序
procedure MyArray(Arr : array of String);

implementation
{$R *.dfm}

procedure MyArray(Arr : array of String);
var
  i : Integer;
begin
  //通过High函数找出此数组的个数并显示其内容值
  for i:= 0 to High(Arr) do
    showmessage('Arr[' + IntToStr(i) + ']=' + Arr[i]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  //传入一个数组类型的参数
  MyArray(['Delphi','is','good']);
end;
```

A.9.4 过程与函数的重载(Overloading)

在同一个有效范围(Scope)中，如果您想以相同的名称同时声明多个子程序，那么您就必须使用 Object Pascal 所提供的重载(Overloading)方法。一个具有重载特性的子程序在声明时必须加上 overload 的编译指引关键字，而且这些子程序的参数行必须不同。

范例 A-10

目的：本范例主要说明各种子程序重载的使用方法。

```
(略)
  //声明各种不同类型的myAdd子程序
  function myAdd(X, Y : Integer):String; overload;
  function myAdd(X, Y : Real):String; overload;
  function myAdd(X, Y, Z : String):String; overload;
  procedure myAdd(X, Y : String); overload;

implementation

{$R *.dfm}
//返回两个整数相加的结果
function myAdd(X, Y : Integer):String; overload;
begin
  Result := IntToStr(X + Y);
```

```
end;
//返回两个实数相加的结果
function myAdd(X, Y : Real):String; overload;
begin
   Result := FloatToStr(X + Y);
end;
//返回三个字符串相加的结果
function myAdd(X, Y, Z : String):String; overload;
begin
   Result := X + Y + Z;
end;
//显示两个字符串相加的结果
procedure myAdd(X, Y : String); overload;
begin
   Showmessage(X + Y);
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
   //调用整数的函数
   Showmessage(MyAdd(10,20));     //30
   //调用实数的函数
   Showmessage(MyAdd(10.1,20.2)); //30.3
   //调用字符串的函数
   Showmessage(MyAdd('10','20', '30')); //102030
   //调用字符串的过程
   MyAdd('10','20'); //在过程中显示1020
end;
```

A.10 程序块

程序块(Block)在 Object Pascal 中是一个非常重要的概念。因为程序块会直接影响变量生命周期。一个块通常会分成声明部分及运行部分。

```
Procedure TForm1.Button1Click(Sender: TObject);
Type            //块开始、声明部分开始
  ……
Const
  ……
Var
  ……
Begin   //声明部分结束、运行部分开始
  ……
```

```
End;    //运行部分结束，块结束
```

上面以 TButton 为例，这整个就是一个块，块中可分为声明与运行两部分。声明中有“Type(类型声明)”、“const(常量声明)”及“var(变量声明)”等的部分。其顺序通常以类型声明为首，为的是让变量及常量可以参照。这些声明部分您可视需要而定。在这个块中所声明的变量、常量或类型会随着块的开始而开始，同样也会随着块的结束而结束。将这个概念给予扩大至单元(unit)及工程(project)。其关系如图 A.9 所示。

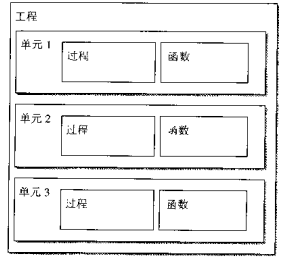


图 A.9　变量的有效范围

在图 A.4 中，可以清楚的看出，在工程中所声明的变量可供“程序单元 1”、“程序单元 2”、“程序单元 3”及“所有程序单元所包含的过程及函数”访问；在程序单元中所声明的变量就只会影响该程序单元内的过程及函数；在过程或函数中所声明的变量就只能在过程或函数中使用。

范例 A-11

目的：本范例主要说明变量在各种不同块(如单元、对象及过程等)中的有效性。

```
Unit Unit1;
  ……略……
Type
  TForm1 = class(TForm)
   ……略……
  private
    { Private declarations }
    //声明一个整数变量，它只能在TFrom对象中使用
    A : Integer;
   ……略……
  end;
var
```

```
  Form1: TForm1;
  //声明一个整数变量，它属于整个 unit
  A:Integer;
  Procedure ShowUnitVar;
Implementation
{$R *.DFM}
procedure ShowUnitVar;
begin
  //因为这个过程是属性单元的，因此在这里找不到变量 A，它会往外找
  //，也就是往 unit 找，单元的变量 A，当时未设置初值，因此其值为 0
  ShowMessage('Show 出 unit 的变量 A,A='+IntToStr(A));
end;
procedure TForm1.Button3Click(Sender: TObject);
var
    A:Integer;
begin
  //在这个小的块中可以找到变量 A，就用这个块的 A
  A:=10;
  ShowMessage('Show 出这个过程中的变量 A,A='+IntToStr(A));
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
   //设置属性 TForm 对象变量 A 的初值
   A:=20;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  //在这个小的块中找不到变量 A，因此往外找，先找 TForm
  //对象，如果找不到，再找 unit
  ShowMessage('Show 出这个 TForm 对象中的变量 A,A='+IntToStr(A));
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  //调用单元的过程
  ShowUnitVar;
end;
end.
```

附 录 B

Object Pascal 程序设计（二）

本 章 目 标

本章将延续附录 A 继续讨论有关其他 Object Pascal 的相关主题。本章所要讨论的主题包括：Object Pascal 的表达式、运算符及 Object Pascal 所提供的数据类型，包括 Simple Types(简单类型)、String Types(字符串类型)、Structured Types(结构类型)、Pointer Types(指针类型)及 Variant Types(变体类型)等。

本 章 主 题

B.1 表达式

B.2 运算符

B.3 Simple Types(简单类型)

B.4 String Types(字符串类型)

B.5 Structured Types(结构类型)

B.6 Pointer Types(指针类型)

B.7 Variant Types(变体类型)

B.1 表达式

在 Object Pascal 最简单的表达式(Expressions)就是变量与常量(请参考附录 A 的详细说明)，如果比较复杂些的表达式就必须通过运算符(Operator)、函数(Function)调用或是通过字符串与数组的索引值存取的方式，甚至是通过强制类型转换的方式所组成。有关运算符在稍后的章节为您说明；函数的调用方式请参考附录 A-9 的详细说明；而字符串与数组的索引值存取方式，笔者也将在稍后的数据类型章节中加以说明。这里笔者将为您说明有关强制类型转换的方式，Object Pascal 所提供的强制类型转换语法如下：

```
TheNewType(Value or Variable)
```

上述语法中，TheNewType 代表所要强制转换的类型，其参数可以是一个数值或是一个变量。如果其参数为数值，那么这就是所谓的“数值类型转换”；反之，如果该参数是变量，那么这就是所谓的“变量类型转换”。“数值类型转换”的新类型与转换的数值必须都是序数类型或是指针类型，下面的范例示范几个常用的类型转换。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;  //整数变量
  c : Char;     //字符变量
  b : Boolean;  //布尔变量
  p : ^Boolean; //布尔的指针变量
  l : LongInt;  //长整数，用于保存内存地址
begin
  //将字符强制转换为整数，其代表 ASCII 码
  i := Integer('A');
  ShowMessage(IntToStr(i)); //65
  //将数字转换字符，其代表由 ASCII 码转为字符
  c := Char(65);
  ShowMessage(c);  //A
  //将整数转换布尔值，0 为 False，非 0 为 True
  b := Boolean(0);
  ShowMessage(BoolToStr(b,true)); //False
  //将一个指针变量指向 b 的内存地址
  p := @b;
  //通过强制类型转换获取内存地址
  l := LongInt(p);
  ShowMessage(IntToStr(l)); //1242555 内存地址
end;
```

“变量类型转换”与“数值类型转换”最大的不同在于前者只会判断两个数据类型所占的内存大小是否相同，并判断表达式确定没有包含实数与整数的混合运算，另外一个不同是：“数值类型转换”不能出现在指定语句的左边，而“变量类型转换”则是被允许的。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  c : char;
begin
  //“变量类型转换”允许放置于指定语句的左边
  ShortInt(c) := 65;
  Showmessage(c); //A
  //“数值类型转换”不允许放置于指定语句的左边
  //ShortInt('A') := 65;
end;
```

B.2 运算符

Object Pascal 的表达式(Expressions)可以是由运算符(Operator)及操作数(operand)所组成的。什么是运算符呢？什么是操作数呢？看了以下的说明就明白了。例如：A:=B+1;这完整的语句就是表达式，而其中的变量 A、变量 B 及数值 1 都被称为操作数，而“+”则称为运算符(也是本小节所要讨论的重点)。因此，我们通过举一反三的归纳可以说，操作数它可以是变量、常量、数值或函数(将返回值当成数值就好了)。事实上，运算符可以分成一元运算符及二元运算符两大类。

```
//一元运算符(只包括一个操作数)
-A                          //将变量A取负值
@A                        //获取变量A的内存位置
NOT  B                  //将布尔变量B反向(True变False，False变True)

//二元运算符(包含两个操作数)
A+B                          //两数相加
(A+B)+(A*B)                  //较复杂的计算
(A-B)>0 and (A>5);    //逻辑判断
```

以下将要介绍的运算符有：算术运算符(Arithmetic Operators)、布尔运算符(Boolean Operators)、逻辑运算符(Logical Operators)、字符串运算符(String Operators)、集合运算符(Set Operators)、关系运算符(Relational Operators)等。

B.2.1 算术运算符(Arithmetic Operators)

算术运算符中可分为一元算术运算符及二元算术运算符。其中一元算术运算符分为-(负号)、+(正号)等运算符，而二元算术运算符分为+(加法)、-(减法)、*(乘法)、\(除法)、div(取商数)及 mod(取余数)等。分述如表 B.1 和表 B.2 所示。

表 B.1 一元算术运算符

运算符	说明	操作数类型	返回值类型	范例	
-	负号	整数、实数	整数、实数	-2	{-2}
+	正号	整数、实数	整数、实数	+3	{3}

表 B.2 二元算术运算符

运算符	说明	操作数类型	返回值类型	范例
+	加法	整数、实数	整数、实数	2+3 {5}
-	减法	整数、实数	整数、实数	2-3 {-1}
*	乘法	整数、实数	整数、实数	2*3 {6}
/	除法	整数、实数	实数	2/3 {0.66..7}
Div	除法取商数	整数	整数	2 div 3 {0}
Mod	除法取余数	整数	整数	2 mod 3 {2}

Object Pascal 的算术运算符包含下列几个重要的特性：

- 当您使用“+(加法)”、“-(减法)”及“*(乘法)”执行算术运算时，当操作数中有一个为实数时，其运算结果则为 Extended 数据类型。
- 假如您使用“/”执行算术的除法运算时，无论其操作数为整数或实数时，其运算结果都为 Extended 数据类型。
- Div 算术运算符将会舍弃“/”运算符计算结果的小数部分，而只返回整数部分。
- 在 Object Pascal 的除法运算中只要是分母为 0，系统就会触发一个 EzeroDivide 异常。这里所谓的除法运算包括：/、Div 与 Mod 等。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  e : Extended;   //声明实数变量
  i, A, B : Integer;  //声明整数变量
begin
  A := 3;   //设置变量初始值
  B := 0;
  try
    //分母与分子皆为数字时，并不会产生异常
    e := 3 / 0;
    Showmessage(FloatTOStr(e));  //INF
    //当分子或分母为变量时，
    //下列几个语句皆会产生 EZeroDivide 的异常
    e := A / B;
    Showmessage(FloatToStr(e));
    i := A div B;
    Showmessage(inttostr(i));
    i := A mod B;
    Showmessage(inttostr(i));
  except
    //E.ClassName 可获取此异常是哪一种类
    //E.Message 可获取此异常的显示信息
    on E: Exception do Showmessage(E.ClassName);
```

```
    end;
end;
```

B.2.2 布尔运算符(Boolean Operators)

Object Pascal 所提供的布尔运算符包括 not、and、or、xor 等，这些运算符可接受任何布尔类型。分述如表 B.3 所示。

表 B.3

运算符	说明	操作数类型	返回值类型	范例
Not	将布尔值反向	Boolean	Boolean	Not 3>5 {True}
And	逻辑 And 运算	Boolean	Boolean	3>5 and 6>3 {False}
Or	逻辑 Or 运算	Boolean	Boolean	3>5 Or 6>3 {True}
Xor	逻辑 Xor 运算	Boolean	Boolean	3>5 Or 6>3 {True}

以下将各个布尔运算符列表说明：

- not 布尔运算符代表否定的意思，它会将布尔值加以反应，如表 B.4 所示。

表 B.4

Not 布尔运算符	Tru e	False
	False	True

- and 布尔运算符就如同“且”的意思，它必须左右两边都为成立(True)，结果才会成立(True)，否则为 False，如表 B.5 所示。

表 B.5

And 布尔运算符	True	False
True	True	False
False	False	False

- or 布尔运算符就如同“或”的意思，它只要有一者成立(True)，结果就会成立(True)，否则为 False，如表 B.6 所示。

表 B.6

Or 运算符	True	False
True	True	True
False	True	False

- xor 布尔运算符就如同“异或”的意思，它只要有一者为成立(True)，另一者不成立(False)，结果才会成立(True)，否则为 False，如表 B.7 所示。

表 B.7

Xor 运算符	True	False
True	False	True
False	True	False

B.2.3 逻辑运算符(*Logical Operators*)

Object Pascal 提供了 not、and、or、xor、shl 及 shr 等六种逻辑运算符。虽然我们已经在 B.2.2 小节中介绍 not、and、or 及 xor 等运算符，但这两个小节所要介绍的运算符所针对的对象是不一样的。在 B.2.2 小节中所介绍的布尔运算符是针对布尔运算，而本小节所要介绍的逻辑运算符则是针对整数的位进行运算，因此两者是完全不一样的。还有一点您必须特别注意，这些运算符的返回值会随着所指定的数据类型而有些差异，这主要是因为不同的数据类型所占的内存大小也不相同。以下范例将以 Byte 类型为例，Byte 类型占有 1 个字节，因此将以 8 个位来表示其二进制内容值，如表 B.8 所示。

表 B.8

运算符	说明	操作数类型	返回值类型	参考范例 B-1
Not	将位值反向	integer	integer	Not 5 {250}
And	对位值做逻辑 and 的运算	integer	integer	5 and 3 {1}
Or	对位值做逻辑 or 的运算	integer	integer	5 or 3 {7}
Xor	对位值做逻辑 xor 的运算	integer	integer	5 xor 3 {6}
Shl	将位左移	integer	integer	5 shl 3 {40}
Shr	将位右移	integer	integer	5 shr 3 {0}

范例 B-1

目的：本范例主要通过五个 TButton 对象的 onClick 事件处理过程说明如何使用逻辑运算符。

1. NOT 逻辑运算符

```
procedure TForm1.ButtonNOTClick(Sender: TObject);
var
  A:byte;
begin
  //先声明变量 A 为 Byte 类型，因为数据类型的不同会有不同结果
  A:= 5;
  //显示结果，其结果为 250
  showmessage(inttostr(not A));
end
```

2. AND 逻辑运算符

```
procedure TForm1.ButtonANDClick(Sender: TObject);
var
  A,B:byte;
begin
  //先声明变量A,B为Byte类型，因数据类型不同会有不同的结果
  A:= 5;
  B:= 3;
  //显示结果，其结果为1
  showmessage(inttostr(A and B));
end;
```

3. OR 逻辑运算符

```
procedure TForm1.ButtonORClick(Sender: TObject);
var
  A,B:byte;
begin
  //先声明变量A,B为Byte类型，因数据类型不同会有不同的结果
  A:= 5;
  B:= 3;
  //显示结果，其结果为7
  showmessage(inttostr(A or B));
end;
```

4. XOR 逻辑运算符

```
procedure TForm1.ButtonXORClick(Sender: TObject);
var
  A,B:byte;
begin
  //先声明变量A,B为Byte类型，因数据类型不同会有不同的结果
  A:= 5;
  B:= 3;
  //显示结果，其结果为6
  showmessage(inttostr(A xor B));
end;
```

5. SHL 逻辑运算符

```
procedure TForm1.ButtonSHLClick(Sender: TObject);
var
  A,B:byte;
begin
  //先声明变量A,B为Byte类型，因数据类型不同会有不同的结果
  A:= 5;
  //将A的内容值左移3位，A = 00000101左移3位之后的内容为
  //00101xxx，其中的xxx系统会将其补0。
```

```
  B:= 3;
  //显示结果，其结果为 40
  showmessage(inttostr(A shl B));
end;
```

6. SHR 逻辑运算符

```
procedure TForm1.ButtonSHRClick(Sender: TObject);
var
  A,B:byte;
begin
  //先声明变量 A,B 为 Byte 类型，因数据类型不同会有不同的结果
  A:= 5;
  //将 A 的内容值右移 3 位，A = 00000101 右移 3 位之后的内容为
  //xxx00000，其中的 xxx 系统会将其补 0。
  B:= 3;
  //显示结果，其结果为 0
  showmessage(inttostr(A shr B));
end;
```

以下就 not 逻辑运算符进行说明。因为我们声明 A 为 Byte 类型，Byte 是一种没有正负号且其所占内存大小为 1 Byte，因此它就有 8 Bits。若将整数 5 以 8 bits 来表示，它会变成 00000101。若将它每个位加以反向(Not)，则其反向的结果为 11111010，您若将 11111010 换算成十进制，则它就是 250 了。其余的逻辑运算符您可以利用这个方法加以类推即可得知其计算结果。

B.2.4 字符串运算符(String Operators)

Object Pascal 所提供的字符串运算符非常简单，只有一个“+”运算符，如表 B.9 所示。

表 B.9

运算符	说明	操作数类型	返回值类型	范例
+	将字符串或字符加以连接	String,char	String	'abc'+'d' {'abcd'}

“+”运算符的运算结果和任何字符串类型都是兼容的。但当两个操作数都是短字符串或字符类型，其最后连接的长度只能保留 255 字符，超过的部分将会被舍弃。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i , L : Integer;
  s : ShortString;
begin
  //获取 ShortString 所占内存的大小
  Showmessage(IntToStr(SizeOf(S))); //256
  s := '';
  For i := 1 to 300 do s := s + 'A';
```

```
  L := Length(s); //获取 S 字符串的长度
  Showmessage(InttoStr(L)); //255
end;
```

另外，Delphi 在 System 单元文件中提供另一个与“+”运算符功能相同的 Concat 函数。Concat 函数与“+”运算符一样都可以将指定的字符串加以连接，但是“+”运算符比 Concat 函数的处理速度来得快。

B.2.5 集合运算符(Set Operators)

Object Pascal 所提供的集合运算符包括：联结(+)、差集(-)、交集(*)、包含于(<=)、包含(>=)、等于(=)、不等于(<>)及属于(in)等。以下范例笔者将以集合 A、B 来做运算。集合 A 的内容为[1,2,3]、B 的内容为[2,3,4]，如表 B.10 所示。

表 B.10

运算符	说明	操作数类型	返回值类型	参数范例 B-2
+	联结	集合类型	集合类型	A+B {1,2,3,4}
-	差集	集合类型	集合类型	A-B {1}
*	交集	集合类型	集合类型	A*B {2,3}
<=	包含于	集合类型	Boolean	A<=B {False}
>=	包含	集合类型	Boolean	A>=B {False}
=	等于	集合类型	Boolean	A=B {False}
<>	不等于	集合类型	Boolean	A<>B {True}
In	属于	序数类型、集合类型	Boolean	3 in B {True}



B-2

目的：本范例主要说明如何使用集合运算符。

```
procedure TForm1.Button1Click(Sender: TObject);
type
  MySetType = set of Byte;
var
  A,B:MySetType;
begin
  A:=[1,2,3];
  B:=[2,3,4];
  //说明 "<=" 的用法
  if A<=B then Showmessage('A<=B is True')
  else ShowMessage('A<=B is False');
  //说明 ">=" 的用法
  if A>=B then Showmessage('A>=B is True')
```

```
    else ShowMessage('A>=B is False');
    //说明 "=" 的用法
    if A=B then Showmessage('A=B is True')
    else ShowMessage('A=B is False');
    //说明 "<>" 的用法
    if A<>B then Showmessage('A<>B is True')
    else ShowMessage('A<>B is False');
    //说明 "in" 的用法
    if 3 in B then Showmessage('3 in B is True')
    else ShowMessage('3 in B is False');
end;
```

B.2.6 关系运算符(Relational Operators)

Object Pascal 所提供的关系运算符包括：等于(=)、不等于(<>)、小于(<)、大于(>)、小于等于(<=)、大于等于(>=)等，分述如表 B.11 所示。

表 B.11

运算符	说明	操作数类型	返回值类型	范例
<	小于	Simple,string	Boolean	1 < 3 {True}
>	大于	Simple,string	Boolean	1 > 3 {False}
<=	小于等于	Simple,string, pchar	Boolean	1 <= 3 {True}
>=	大于等于	Simple,string, pchar	Boolean	1 >= 3 {False}
=	等于	Simple,string, pchar	Boolean	1 = 3 {False}
<>	不等于	Simple,string, pchar	Boolean	1 <> 3 {True}

B.2.7 类运算符(Class Operators)

Object Pascal 提供 is 与 as 两个类运算符。is 运算符负责进行动态数据类型的检查，而 as 运算符则负责进行强制类型转换。语法如下：

```
//is 运算符的语法
Object is ClassType;  //其返回值为 True 或 False
//as 运算符的语法
Object as NewClassType; //返回新的类类型
```

当 as 运算符所指定的新类类型无法成功转换时，系统会触发一个 EinvalidCast 的异常。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    //尝试将触发 Button1Click 事件的对象转为 TEdit 对象
```

```
      with (sender as TEdit) do Showmessage(Text);
    except
      //正常的情况，Sender 会等于 TButton 对象本身，
      //因此会产生 EInvalidCast 异常
      On E: Exception do Showmessage(E.ClassName);
    end;
  end;
```

范例 B-3

目的：本范例主要说明如何使用 is 和 as 等类运算符以获取 TForm 对象上的所有对象的标题名称，在这个范例中，笔者分别加入三个 TEdit 与 TButton 对象。

```
procedure TForm1.Button3Click(Sender: TObject);
var
   I:Integer;
   MyString:String;
begin
   MyString:='TForm 对象上的对象清单为: '+ #10;
   //利用 ComponentCount 获取 TForm 对象的控制组件个数
   For I:=0 to ComponentCount -1 do
      //根据此对象的类型加以分别处理
      //如果是 TButton 对象则获取其 Caption 属性值
      if Components[I] is TButton then
        MyString:= MyString + TButton(Components[I]).Caption + #10
      //如果是 TEdit 对象则获取其 Text 属性值
      else if Components[I] is TEdit then
        MyString:=MyString+ TEdit(Components[I]).Text + #10;
   //show 出这些对象的 Name 属性
   ShowMessage(MyString);
end;
```

B.3 Simple Types(简单类型)

Object Pascal 内建的数据类型简单来说可分为 simple(简单类型)、String(字符串类型)、Structured(结构类型)、Pointer(指针类型)、procedural(过程类型)及 variant(变体类型)等。在正式介绍数据类型之前，先为大家介绍一个 SizeOf 函数，这个函数可以返回某个内置数据类型或自定义类型所占内存的大小。

范例 B-4

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  myString:String;
begin
  myString:= 'Integer 占'+IntToStr(SizeOf(Integer))+'Bytes'+ #10;
  myString:= myString + 'String 占' +IntToStr(SizeOf(String))
             +'Bytes'+#10;
  myString:= myString + 'Boolean 占'+IntToStr(SizeOf(Boolean))
             +'Bytes'-#10;
  ShowMessage(myString);
end;
```

运行结果如图 B.1 所示。



图 B.1 SizeOf 函数

在 Object Pascal 的 Simple 数据类型中，可分为序数(Ordinal)与实数(Real)两大类。这两者最大的不同是属于序数的数据类型可以使用序数函数(Ordinal Routines)。序数数据类型又可分为整型类型(Integer)、字符类型(character)、布尔类型(Boolean)、枚举类型(enumerated)及子界类型(Subrange)。在开始说明简单数据类型之前，先来谈谈与序数相关的函数，如表 B.12 所示。

表 B.12

序数函数	参数	返回值
Ord	序数表达式	运算结果的顺序
Pred	序数表达式	运算结果的前行者(Predecessor)
Succ	序数表达式	运算结果的后继者(Successor)
High	序数数据类型或变量	指定数据类型中的最高的值
Low	序数数据类型或变量	指定数据类型中的最低的值

范例 B-5

目的：本范例主要说明序数函数 Ord 的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
type
  MyColors = (RED,BLUE,GREEN);
var
```

```
  MyString1,
  MyString2,
  MyString3: string;
begin
  MyString1 := 'BLUE 的次序值(ordinal value)是'+IntToStr(Ord(BLUE)) +
               #10;
  MyString2 := '字母 A 的 ASCII code 值是'+IntToStr(Ord('A'))+ #10;
  MyString3 := '字母 a 的 ASCII code 值是'+IntToStr(Ord('a'));
  Showmessage(MyString1+MyString2+MyString3);
end;
```

运行结果如图 B.2 所示。



图 B.2 Ord 序数函数

范例 B-6

目的：本范例主要说明序数函数 Pred 与 Succ 的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
type
  MyColors = (RED,BLUE,GREEN);
var
  MyString1,
  MyString2,
  MyString3: string;
begin
  MyString1 := '数字 9 的前一个数是: ' + IntToStr(Pred(9)) + #10;
  MyString2 := '数字 8 的后一个数是: ' + IntToStr(Succ(8)) + #10;
  if Pred(GREEN) = BLUE then
    MyString3 := MyString3 + 'BLUE 是排在 GREED 之前';
  ShowMessage(MyString1+MyString2+MyString3);
end;
```

运行结果如图 B.3 所示。



图 B.3 Pred 与 Succ 序数函数

范例 B-7

目的：本范例主要说明序数函数 High 与 Low 的使用方法。

```
procedure TForm1.Button1Click(Sender: TObject);
const
   PI = 3.1416;
var
  //声明数组
  MyArray: array[5..38] of Double;
  //声明一个 Word 整数变量
  I: Word;
  //声明一个字符串变量
  MyString: string;
begin
  //设置两个数组的值
  for I := Low(MyArray) to High(MyArray) do
     MyArray[I] := I * PI;
  //调用汇总的子程序，汇总 MyArray 的值
  Str(Total(MyArray):10:4, MyString);
  ShowMessage('MyArray 的总合为: '+MyString);
end;
//自定义函数，用来计算数组的总和
function Total( var ArrayX: array of Double): Double;
var
  I: Word;
  R: Real;
begin
  R := 0;
  for I := 0 to High(ArrayX) do R := R + ArrayX[I];
  Total := R;
end;
```

运行结果如图 B.4 所示。



图 B.4 High 与 Low 序数函数

B.3.1 Integer Types(整数数据类型)

Object Pascal 的整型类型是所有整型类型的一个通称，它可包含 Integer、Cardinal、

ShortInt、SmallInt、Int64、Byte、Word 及 LongWord 等。在 Delphi 官方中建议尽量的使用 Integer 与 Cardinal 两个类型的整数，因为这两者的运算速度比较快，如表 B.13 所示。

表 B.13

类型	范围	格式
Integer	-2147483648~2147483647	Signed 32 bit
Cardinal	0~4294967295	Unsigned 32 bit
ShortInt	-128~127	Signed 8 bit
SmallInt	-32768~32767	Signed 16 bit
LongInt	-2147483648~2147483647	Signed 32 bit
Int64	-2^{63}~2^{63}-1	Signed 64 bit
Byte	0~255	Unsigned 8 bit
Word	0~65535	Unsigned 16 bit
LongWord	0~4294967295	Unsigned 32 bit

声明及使用方式如下：

```
Var
   I : Integer;
Begin
   I:=0;
   I:=I+1;
End;
```

B.3.2 Character Types(字符类型)

Object Pascal 所提供的字符数据类型包括 Char、AnsiChar 及 WideChar 等，如表 B.14 所示。

表 B.14

类型	格式
Char	8 bit
AnsiChar	8 bit
WideChar	16 bit

一般所指的字符类型(Characte Types)就是 Char，Char 与 AnsiChar 是相同的，这两者就是大家所熟知的 ASCII。而 WideChar 的前 256 个字符与 AnsiChar 是相同的，通常它被用于 Unicode 字符。以下就声明及使用方式加以说明：

范例 B-8

```
Procedure TForm1.Button1Click(Sender: TObject);
Var
   MyChar : Char;  //声明
Begin
   MyChar := 'A';    //设置初值
   MyChar := succ(MyChar);
   //显示目前MyChar内容的ASCII码
   ShowMessage(MyChar+'-'+IntToStr(Ord(MyChar))); //B=66
end;
```

B.3.3 Boolean Types(布尔类型)

布尔类型主要用于程序流程或运行结果的判断，换句话来说，不是 YES 就是 NO。布尔类型其主要的值就是 True 与 False。在 Object Pascal 中提供了 Boolean、ByteBool、WordBool 及 LongBool 等四种的布尔类型，您可以配合不同的场合使用，其中所使用的字节最少的是 Boolean，由于这些布尔类型对于 True 及 False 的定义有些微的不同，表 B.15 将其差异加以说明。

表 B.15

Boolean	ByteBool 、WordBool、LongBool
False < True	False <> True
Ord(False) = 0	Ord(False) = 0
Ord(True)=1	Ord(True) <> 0
Succ(False) = True	Succ(False) = True
Pred(True)=False	Pred(False) = True

范例 B-9

```
procedure TForm1.Button1Click(Sender: TObject);
var
    MyBool:ByteBool;
begin
   MyBool:=True;
   if Pred(MyBool) = True then
     ShowMessage('True')
   else
     ShowMessage('False');
end;
```

B.3.4 Enumerated Types(枚举类型)

枚举类型通常包含一组顺序(Order)的数据集合。要声明一个枚举类型，其语法如下：

```
Type 枚举类型名称 = (枚举值 1,枚举值 2,...,枚举值 n);
```

例如您要声明一个与颜色相关的枚举类型，您可以这么做。

```
Type TmyColor = ( RED, BLUE, GREEN, BLACK);
```

声明只是告诉计算机，让计算机认识有这种数据类型而已，如果您想使用它，请参考范例。

范例 B-10

步骤 1 建立新的工程。

步骤 2 在 TForm 对象中加入四个 TRadioButton 对象及一个 TButton 对象，其相关属性设置如表 B.16 所示。

表 B.16

对象名称	属性名称	属性设置值
TradioButton	Name	RadioButtonRED
	Caption	红色
TradioButton	Name	RadioButtonBLUE
	Caption	蓝色
TradioButton	Name	RadioButtonGREEN
	Caption	绿色
TradioButton	Name	RadioButtonBLACK
	Caption	黑色
Tbutton	Name	Button1
	Caption	您所选的颜色是

步骤 3 在单元文件(Unit1)中声明一个“自定义的枚举类型”，并声明其对应的变量。

```
Type
  //声明自定义的枚举类型
  TmyColor = ( RED, BLUE, GREEN, BLACK);

var
  Form1: TForm1;
  //定义 MyColor 为 TMyColor 枚举类型
  MyColor:TMyColor;
```

步骤 4 在 RadioButtonRED 对象的 onClick 事件处理过程中加入以下代码：

```
procecure TForm1.RadioButtonREDClick(Sender: TObject);
begin
```

```
    //将 MyColor 设为 RED
    MyColor := RED;
end;
```

步骤 5 在 RadioButtonBLUE 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.RadioButtonBLUEClick(Sender: TObject);
begin
    //将 MyColor 设为 BLUE
    MyColor := BLUE;
end;
```

步骤 6 在 RadioButtonGREEN 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.RadioButtonGREENClick(Sender: TObject);
begin
    //将 MyColor 设为 GREEN
    MyColor := GREEN;
end;
```

步骤 7 在 RadioButtonBLACK 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.RadioButtonBLACKClick(Sender: TObject);
begin
    //将 MyColor 设为 BLACK
    MyColor := BLACK;
end;
```

步骤 8 在 TButton 对象的 onClick 事件处理过程中加入以下的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var Str1 : String;
begin
   case MyColor of
       RED   : Str1 :='红色';
       BLUE  : Str1 :='蓝色';
       GREEN : Str1 :='绿色';
       BLACK : Str1 :='黑色';
   end;
   //显示出您所选择的颜色
   ShowMessage('您所选的颜色是:'+Str1);
end;
```

步骤 9 保存工程。

步骤 10 按【F9】功能键执行工程。工程运行后，您可以选择某个 TRadioButton 对象，再按下 TButton 对象，此时系统会响应您所选择的颜色，如图 B.5 所示。



图 B.5 枚举类型范例

B.3.5 SubRange Types(子界类型)

子界类型主要提供用户自定义变量的范围，通常它是一个主要范围中的一段连续的小部分，您只要说明其最小值及最大值即可，您可以针对整数、字符或枚举类型设置子界类型，可参考范例 B-11。

范例 B-11

```
Type MyNumber = -128..127;
Type MyChar = 'A'..'Z';
//要使用枚举类型的自定义范围，必须要先有声明才行，倘若 Tcolors
//没有声明，则 TmyColor 会出错
TColors  = (Red,Blue,Green,Yellow, Orange ,Purple,White,Black);
TMyColor = Green..White;
```

B.3.6 Real Types(实数类型)

实数类型的变量与整型类型最大的不同在于实数包含了小数部分，这种数据类型为了精确度上的考虑，因而使用浮点数来记录数据，因此在做实际值可能会有一点点的误差。因此在写程序时，尽量避免用实数类型来作为循环的判断条件。Object Pascal 所提供的实数类型包含如表 B.17 所示的几种。

表 B.17

类型	范围	有效位数	内存
Real48	$2.9*10^{-39}..1.7*10^{38}$	11~12	6 Bytes
Single	$1.5*10^{-45}..3.4*10^{38}$	7~8	4 Bytes
Double	$5.0*10^{-324}..1.7*10^{308}$	15~16	8 Bytes
Extended	$3.6*10^{-4951}..1.1*10^{4932}$	19~20	10 Bytes
Comp	$-2^{36}+1..2^{63}-1$	19~20	8 Bytes
Currency	-922337203685477.5808 .. 922337203685477.5807	19~20	8 Bytes
Real	$5.0*10^{-324}..1.7*10^{308}$	15~16	8 Bytes

B.4 String Types(字符串类型)

字符串(String)这种数据类型，几乎是写程序最常用到的，诸如：信息框、对象的 Caption 属性或对象的 Name 属性。一般来说您可以将字符串当作是多个字符(Char)所组合起来的一种数据类型。Object Pascal 所提供的字符串数据类型至少有表 B.18 所示的三种。

表 B.18

类型	最大长度	内存需求	使用对像
ShortString	255 个字符	2 至 256 Bytes	一般情况
AnsiString	2^{31} 个字符	4 Bytes 至 2GB	8-Bit(ANSI)字符
WideString	2^{30} 个字符	4 Bytes 至 2GB	Unicode 字符或 COM Server and Interfaces

一般来说，字符串的比较大小是以个别字符作为比较基础，例如'ABC' > 'AB' > 'A' > ''。所以当字符串长度为 0 时为最小，如表 B.19 所示。

表 B.19

字符串值	显示结果
'Delphi'	Delphi
'It's good'	It's good
'' ''	"可以使用 ASCII(39)来代替，如 chr(39)
' '	一个空格符
''	空字符串
#10	换行(#后面接着 ASCII)

范例 B-12

目的：本范例将示范如何对一个字符串中的个别字符做处理。

```
procedure TForm1.Button1Click(Sender: TObject);
var
   myString :String;
   I:Integer;
begin
  myString := 'AEPDKIWSDSESLWXDAQADGERQ';
  For I:=1 to Length(myString) do
     if I mod 2 = 0 then begin //将单数字符替换成'*'
       MyString[I]:= '*';
     end;
  ShowMessage(myString);
end;
```

B.5 Structured Types(结构类型)

简单地说，结构类型，它是一个可以允许用户将其他已知的数据类型加以组合起来。这句话的意思就是说，您除了可以将上述所提的 Integer、Boolean 及 Character 等的数据类型加以组合外，更可将已经定义过的“结构类型”加以组合成一个新的“结构类型”。Object Pascal 所提供的结构类型可分为 Set、Array、Record、File、Class、Class Reference 及 Interface 等。以下就几种较常使用的类型加以说明：

B.5.1 Set Types(集合类型)

集合类型是由一群相同序数类型的元素所组合的集合，其序数介于 0~255 之间。通常使用 Set of 保留字来声明，声明语法如下：

```
Set of 集合类型名称
```

以下将以实际范例来说明如何声明集合数据类型。

```
Type
    //声明一个集合类型，其值由 0~255
    TmyIntset = 0..255
    //Tintset 1 主要是继承 TmyIntset 而来，同样具有 0~255 的值
    Tintset1 = Set of TmyIntset
```

范例 B-13

```
procedure TForm1.Button1Click(Sender: TObject);
var
   //声明一个范围
   MySet : set of 'a'..'d';
   MyChar : Char;
Begin
   //设置真正的范围
   MySet := ['a','b','c'];
   MyChar := 'a';
   //判断某个字符是否包含在集合变量内
   if MyChar in MySet then
     ShowMessage(MyChar +' in MySet')
   else
     ShowMessage(MyChar +' not in MySet');
end;
```

B.5.2 Array Types(数组类型)

数组类型通常是将相同类型且数量大小固定的数据类型加以集合在一起，但 Object

Pascal 除了提供固定大小的数组外(Static Arrays)，另提供了可动态增加大小的动态数组(Dyanmic Arrays)。其声明都以 Array 这个保留字来声明，其声明语法如下：

数组名=Array[范围] of 数据类型名称



B-14

目的：本范例主要说明 Static Array 的一维及二维数组

1. 一维数组数据类型

```
procedure TForm1.Button1Click(Sender: TObject);
Type
  //声明一个数组类型
  TmyArray = array[0..9] of integer;
var
    //定义一个数组变量
    MyArray:TMyArray;
    I:Integer;
    myString:String;
begin
    //给予数组值，并显示
    for i:=0 to 9 do
    begin
      myArray[i]:=i;
      myString:=MyString+ IntToStr(myArray[i]) + '  ';
    end;
    ShowMessage(myString);
end;
```

2. 二维数组数据类型

```
procedure TForm1.Button2Click(Sender: TObject);
Type
  //声明一个数组类型
  TmyArray = array[0..9,0..9] of integer;
var
    //定义一个数组变量
    MyArray:TMyArray;
    I,J:Integer;
    myString:String;
begin
    //给予数组值，并打印
    for i:=0 to 9 do
    begin
        for j:=0 to 9 do
```

```
        begin
          myArray[i,j]:=i*10+j;
          myString:=MyString+IntToStr(myArray[i,j])+'  ';
        end;
        myString:=myString + #10;
    end;
    ShowMessage(myString);
end;
```

运行结果如图 B.6 所示。

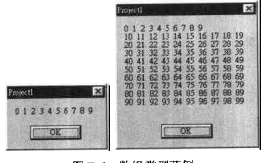


图 B.6　数组类型范例

B-15

目的：本范例目的是说明如何使用 Dynamic Array。

```
procedure TForm1.Button1Click(Sender: TObject);
var
    //定义一个变动数组变量
    MyArray: array of Integer;
    I:Integer;
    myString:String;
begin
    //设置变动数组的大小，长度同样要在 0~255 之间
    SetLength(MyArray,10);
    //给予数组值，并显示
    for i:=1 to 9 do
    begin
      myArray[i]:=i;
      myString:=MyString+ IntToStr(myArray[i]) + '  ';
    end;
    ShowMessage(myString);
end;
```

B.5.3　*Record Types*(记录类型)

在数组中主要是存放同一数据类型的数据，如果您需要将不同数据类型的数据加入集

合，您可以使用记录类型(Record Types)。其声明都以 Record 这个保留字来声明。



B-16

目的：本范例目的是说明如何使用记录类型(Record Types)

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
   Dialogs, StdCtrls;
type
  //声明年月日为记录类型
  TMyDate = record
    DD: 1..31;
    MM: 1..12;
    YYYY: 1900..1999;
  end;
  //定义学生的数据类型
  TMyStudent = record
      Name : String[50];
      IDNO : String[20];
      BirthDay:TMyDate;
  end;
  //定义一个小班制的学生总集合
  TStudents = array[0..50] of TMyStudent;
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    EditName: TEdit;
    EditIDNO: TEdit;
    EditYear: TEdit;
    Button1: TButton;
    Label4: TLabel;
    Label5: TLabel;
    EditDay: TEdit;
    ComboBoxMonth: TComboBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  //用来记录目前是第一位学生
  Flag:Integer;
  //用来记录学生的数据
  Students : TStudents;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
var
  Str:String;
begin
  //将学生的数据存到 Record
  with Students[Flag] do
  begin
    Name:=EditName.Text;
    IDNo:=EditIDNO.Text;
    BirthDay.DD:=StrToInt(EditDay.Text);
    Birthday.MM:=StrToInt(ComboBoxMonth.Text);
    BirthDay.YYYY:=StrToInt(EditYear.Text);
  end;
  //将学生的数据从 Record 读出来
  with Students[Flag] do
  begin
    Str:='姓名:'+Name+#10;
    Str:=Str+'学号:'+IDNO+#10;
    Str:=Str+'生日年:'+IntToStr(BirthDay.YYYY)+#10;
    Str:=Str+'生日月:'+IntToStr(BirthDay.MM)+#10;
    Str:=Str+'生日日:'+IntToStr(BirthDay.DD)+#10;
  end;
  ShowMessage('第'+IntToStr(Flag)+'条数据'+#10+Str);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  //设置初值
  Flag:=0;
end;
end.
```

运行结果如图 B.7 所示。



图 B.7　记录类型范例

B.6　Pointer Types(指针类型)

指针类型，它是一组内存地址的数字，其记录着真正存放数据的内存地址。指针类型通常用于动态产生的对象或变量，所谓动态，就是在程序运行时才产生的对象。例如范例 A-16 就并非动态产生对象或变量，因为程序一开始就指明该班学生为 50 人，因此如果学生少于 50 人，那数据存取当然没问题，若超过 50 人，则无法继续存储数据。如果您的数据条数是无法确定的，那您可能需要使用指针类型会比较恰当一些。

范例 B-17

目的：本范例目的是说明如何使用指针类型(Pointer Types)，本范例修改自范例 B-16 故只列出部分代码。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str:String;
begin
  //创建新的对象
  NEW(Students);

  //将学生的数据存到 Record
  Students^.Name:=EditName.Text;
  Students^.IDNo:=EditIDNO.Text;
  Students^.BirthDay.DD:=StrToInt(EditDay.Text);
  Students^.Birthday.MM:=StrToInt(ComboBoxMonth.Text);
  Students^.BirthDay.YYYY:=StrToInt(EditYear.Text);

  //将学生数据由 Record 取出来
  Str:='姓名:'+Students^.Name+#10;
  Str:=Str+'学号:'+Students^.IDNO+#10;
  Str:=Str+'生日年:'+IntToStr(Students^.BirthDay.YYYY)+#10;
  Str:=Str+'生日月:'+IntToStr(Students^.BirthDay.MM)+#10;
  Str:=Str+'生日日:'+IntToStr(Students^.BirthDay.DD)+#10;
```

```
  ShowMessage(Str);
end;
```

B.7 Variant Types(变体类型)

Object Pascal 提供一种数据类型，系统允许它能自动在各种数据类型间转换，所以被称为变体类型。但事实上天下无白吃的午餐，变体类型可以让用户很方便的使用，但您可能会付出一点点处理时间的代价。

```
procedure TForm1.Button1Click(Sender: TObject);
Var
     MyVariant : Variant;
Begin
     MyVariant := True;                    //此时为布尔类型
     MyVariant := '樱桃小丸子';   //此时为字符串类型
     MyVarinat := 11;                      //此时为数字类型
End;
```

Object Pascal 提供了一组 Variant 数据类型的处理函数，您可以在附录 E.4 小节中找到它。这些函数包括如何创建一个 Variant 数组及多个处理函数。

附 录 C

Delphi 6 常用文件处理函数

本 章 目 标

本章将为大家介绍 Delphi 6 常用的文件处理函数，这些函数可分为目录处理、文件名、文件属性及文件处理函数、文本文件处理函数、类型文件(Typed File)处理函数、无类型文件(Untyped File)处理函数等。

本 章 主 题

C.1 目录处理函数

C.2 文件名处理函数

C.3 文件属性处理函数

C.4 文件处理函数

C.5 文本文件处理函数

C.6 类型文件(Typed File)处理函数

C.7 无类型文件(Untyped File)处理函数

C.1 目录处理函数

本小节将为您介绍 Delphi 6 所提供的目录处理函数，这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先行说明每一个目录处理函数所代表的意义，如表 C.1 所示。

表 C.1

函数名称	单元文件	所代表的意义
ChDir	System	ChDir 函数可更改当前的工作目录，此函数并无返回值，您必须通过 IOResult 函数返回值判断目录是否已成功更改
GetDir	System	GetDir 函数可返回指定驱动器的当前工作目录。驱动器号分别为 0、1、2、3 等。其中 0 代表系统当前的驱动器号、1 代表 A 磁盘、2 代表 B 磁盘、3 代表 C 磁盘，依次类推。此函数无法获取运行状态为成功或失败，假如指定的驱动器号不存在，系统将返回当前驱动器的工作目录
MkDir	System	MkDir 函数可创建一个指定名称的目录，此函数并无返回值，您必须通过 IOResult 函数返回值判断目录是否成功的创建
CreateDir	SysUtils	CreateDir 函数可创建一个指定名称的目录，此函数的返回值为布尔值，当返回值为 True 表示成功创建，反之，则代表创建失败
ForceDirectories	SysUtils 或 FileCtrl	ForceDirectories 函数可创建一个指定路径的目录，此函数的返回值为布尔值。当返回值为 True 表示成功创建，反之，则代表创建失败。ForceDirectories 与 MkDir 和 CreateDir 最大的不同在于后两者的函数只能创建一层的目录，而 ForceDirectories 函数可以创建一个完整路径所需要的所有子目录
RmDir	System	RmDir 函数可删除指定的空目录，也就是此函数只能删除不含任何子目录或文件的目录。此函数的返回值为布尔值。当返回值为 True 表示成功删除，反之，则代表删除失败
RemoveDir	SysUtils	RemoveDir 函数可删除指定的空目录，也就是此函数只能删除不含任何子目录或文件的目录。此函数的返回值为布尔值。当返回值为 True 表示成功删除，反之，则代表删除失败
DirectoryExist	SysUtils 或 FileCtrl	DirectoryExist 函数可判断指定的目录是否存在。此函数的返回值为布尔值。当返回值为 True 表示指定的目录存在，反之，则代表不存在
SetCurrentDir	SysUtils	SetCurrentDir 函数可设置当前的工作目录。此函数的返回值为字符串。当返回值为 True 表示工作目录设置成功，反之，则代表不成功
GetCurrentDir	SysUtils	GetCurrentDir 函数可获取当前的工作目录。此函数的返回值为字符串类型，其代表工作目录的完整路径
DiskSize	SysUtils	DiskSize 函数可获取指定驱动器的空间大小，其计算单位为字节(bytes)。驱动器号分别为 0、1、2、3 等。其中 0 代表系统当前的驱动器号、1 代表 A 磁盘、2 代表 B 磁盘、3 代表 C 磁盘，依次类推。此函数的返回值为整型类型，其代表指定磁盘的字节大小

续上表

函数名称	单元文件	所代表的意义
DiskFree	SysUtils	DiskFree 函数可获取指定驱动器的剩余空间大小，其计算单位为字节(bytes)。驱动器号分别为 0、1、2、3 等。其中 0 代表系统当前的驱动器号、1 代表 A 磁盘、2 代表 B 磁盘、3 代表 C 磁盘，依次类推。此函数的返回值为整型类型，其代表指定磁盘的剩余空间字节

以下将以范例的方式说明每一个函数的使用方法。

ChDir(更改当前的工作目录)

引用单元：System

函数声明：procedure ChDir(S: string);

范例 C-1

```
procedure TForm1.Button1Click(Sender: TObject);
var
  DirName:String;
begin
  ChDir('c:\Winnt');
  if IOResult <> 0 then
    ShowMessage('指定目录无法切换')
  else
    ShowMessage('目录成功的切换');
end;
```

GetDir(获取指定驱动器的目录)

引用单元：System

函数声明：procedure GetDir(Drive: Byte; var S: string);

参数说明：当 Drive 为 0 代表当前的磁盘

当 Drive 为 1 代表 A 磁盘

当 Drive 为 2 代表 B 磁盘

以此类推，如果指定的 Drive 代号不合法则返回—1。

范例 C-2

```
procedure TForm1.Button1Click(Sender: TObject);
procedure TForm1.Button1Click(Sender: TObject);
var
   DriveName:String;
begin
  GetDir(0,DriveName); {当前的磁盘}
```

```
    ShowMessage('当前的目录:'+DriveName);
end;
```

- *MkDir(创建子目录)*

引用单元：System
函数声明：procedure MkDir(S: string); procedure MkDir(P: PChar);



C-3

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MkDir('c:\Test\123\456');
  if IOResult <> 0 then
    ShowMessage('指定目录无法创建')
  else
    ShowMessage('目录成功的创建');
end;
```

- *CreateDir(创建子目录)*

引用单元：SysUtils
函数声明：function CreateDir(const Dir: string): Boolean;



C-4

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if CreateDir('c:\Test3') then
    ShowMessage('目录成功的创建')
  else
    ShowMessage('指定目录无法创建');
end;
```

- *ForceDirectories(依照指定路径创建所有的子目录)*

引用单元：SysUtils 或 FileCtrl
函数声明：function ForceDirectories(const Dir: string): Boolean;



C-5

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ForceDirectories('c:\Test\123\456\789\0123') then
    ShowMessage('目录成功的创建')
```

```
  else
    ShowMessage('指定目录无法创建');
end;
```

- *RmDir(删除空的字目录)*

引用单元：System

函数声明：procedure RmDir(const S: string);
procedure RmDir(P: PChar);

范例 C-6

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  rmDir('c: Test');
  if IOResult <> 0 then
    ShowMessage('指定目录无法删除')
  else
    ShowMessage('目录成功的删除');
end;
```

- *RemoveDir(删除空的字目录)*

引用单元：SysUtils

函数声明：function RemoveDir(const Dir: string): Boolean;

范例 C-7

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if RemoveDir('c:\Test') then
    ShowMessage('目录成功的删除')
  else
    ShowMessage('指定目录无法删除');
end;
```

- *DirectoryExist(判断目录是否存在)*

引用单元：SysUtils 或 FileCtrl

函数声明：function DirectoryExists(const Directory: string): Boolean;

范例 C-8

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DirectoryExists('c:\Winnt') then
```

```
    ShowMessage('目录存在')
  else
    ShowMessage('指定目录不存在');
end;
```

- *SetCurrentDir(设置工作目录)*

引用单元：SysUtils

函数声明：function SetCurrentDir(const Dir: string): Boolean;

C-9

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if SetCurrentDir('c:\Winnt') then
    ShowMessage('工作目录设置成功')
  else
    ShowMessage('工作目录设置失败');
end;
```

- *GetCurrentDir(获取当前的工作目录)*

引用单元：SysUtils

函数声明：function GetCurrentDir: string;

C-10

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('当前的工作目录为:' + GetCurrentDir)
end;
```

- *DiskSize(获取指定磁盘的空间大小)*

引用单元：SysUtils

函数声明：function DiskSize(Drive: Byte): Int64;

C-11

```
procedure TForm1.Button1Click(Sender: TObject);
var
  vSize : Integer;
begin
  vSize := DiskSize(3); //C 磁盘
  ShowMessage('指定磁盘的大小为:' + IntToStr(vSize) + '字节');
end;
```

DiskFree (获取指定磁盘的剩余空间大小)

引用单元：SysUtils
函数声明：function DiskFree(Drive: Byte): Int64;

例 C-12

```
procedure TForm1.Button1Click(Sender: TObject);
var
  vSize : Integer;
begin
  vSize := DiskFree(3); //C 磁盘
ShowMessage('指定磁盘的剩余空间为:' + IntToStr(vSize) + '字节');
end;
```

C.2 文件名处理函数

本小节将为您介绍 Delphi 6 所提供的文件名处理函数，这些函数的处理对象只是文件名而已，这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先行说明每一个目录处理功能函数所代表的意义，如表 C.2 所示。

表 C.2

函数名称	单元文件	所代表的意义
ChangeFileExt	SysUtils	ChangeFileExt 函数可更改指定文件名的扩展名，其返回值为字符串类型，代表更改后的文件名
ExcludeTrailingBackslash	SysUtils	ExcludeTrailingBackslash 函数可删除指定路径名称的最后一个“\”符号，其返回值为新的路径名称
ExcludeTrailingPathDelimiter	SysUtils	ExcludeTrailingPathDelimiter 函数与 ExcludeTrailingBackslash 函数供有相同的功能，它能删除指定路径名称的最后一个“\”符号，其返回值为新的路径名称
ExpandFileName	SysUtils	ExpandFileName 函数能将文件名加在当前路径，其返回值为字符串类型，其代表完整的路径
ExpandUNCFileName	SysUtils	如果使用于单机上 ExpandUNCFileName 函数与 ExpandFileNam 函数具有相同的功能。但是如果用于网络驱动器上，ExpandUNCFileName 函数会将指定的文件转换为 UNC 路径。例如:E 磁盘是由网络邻居 //carlos/temp 所连接而成的，其所转换的结果为//carlos/temp/File Name.txt
ExtractFileDrive	SysUtils	ExtractFileDrive 函数可返回指定路径的驱动器名称，函数的返回值为字符串类型，其代表驱动器名称

续上表

函数名称	单元文件	所代表的意义
ExtractFileDir	SysUtils	ExtractFileDir 函数可返回指定路径的目录名称，此函数的返回值为字符串类型，其代表目录名称
ExtractFileExt	SysUtils	ExtractFileExt 函数可返回指定路径的扩展名，函数的返回值为字符串类型，其代表扩展名
ExtractFileName	SysUtils	ExtractFileName 函数可返回指定路径的文件名，函数的返回值为字符串类型，其代表文件名
ExtractFilePath	SysUtils	ExtractFilePath 函数可返回指定文件所在的路径，函数的返回值为字符串类型，其代表路径名称。此函数与 ExtractFileDir 函数的不同之处，在于前者包括“\”符号
ExtractRelative Path	SysUtils	ExtractRelativePath 函数可返回但有关联性的路径名称，此函数有 BaseName 与 DestName 两个参数。BaseName 参数代表比较关联性的路径名称，它可以包含文件名，假如不包含文件名，那么它必须以“\”符号结尾。DestName 参数代表被比较的完整文件路径，其必须包含文件名。ExtractRelativePath 函数的返回值为具有关联性的文件名
ExtractShortPathName	SysUtils	ExtractShortPathName 函数可返回 DOS 模式下的 8.3 文件格式路径，例如：C:\Program Files\可能被转为 C:\Progra~1\
IncludeTrailingBackslash	SysUtils	IncludeTrailingBackslash 函数将返回以“\”为结尾的路径名称，此函数无法判断此路径是以文件名作结尾，或是目录名称作结尾
IncludeTrailingPathDelimiter	SysUtils	此函数与 IncludeTrailingBackslash 函数具有相同的功能，其可返回以“\”为结尾的路径名称。与 IncludeTrailingBackslash 函数的不同在于 IncludeTrailingPath Delimiter 支持 multibyte character sets
IsPathDelimiter	SysUtils	IsPathDelimiter 函数可判断路径的指定索引位置是否为“\”符号。字符串的索引位置从 1 起算
MatchesMask	Masks	MatchesMask 函数可判断指定路径是否符合掩码(Mask)条件

以下将以范例的方式说明每一个函数的使用方法：

ChangeFileExt (更改指定文件名的扩展名)

引用单元：SysUtils

函数声明：function ChangeFileExt(const FileName, Extension: string): string;

范例 C-13

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  oldFileName,NewFileName : String;
begin
  oldFileName := 'Test.txt';
  NewFileName := ChangeFileExt(oldFileName,'.bak');
  Showmessage('新的文件名为:' + NewFileName);
end;
```

- *ExcludeTrailingBackslash(删除指定路径名称的最后一个"\"符号)*

引用单元：SysUtils

函数声明：function ExcludeTrailingBackslash(const S: string): string;

C-14

```
procedure TForm1.Button1Click(Sender: TObject);
var
  oldDirName,NewDirName : String;
begin
  oldDirName := 'C:\Temp\123\';
  NewDirName := ExcludeTrailingBackslash(oldDirName);
  Showmessage('转换后的目录名称为:' + NewDirName);
end;
```

- *ExpandFileName(将文件名加在当前路径)*

引用单元：SysUtils

函数声明：function ExpandFileName(const FileName: string): string;



C-15

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileName : String;
begin
  FileName := 'MyFileName.bak';
  FileName := ExpandFileName(FileName);
  Showmessage(FileName);
end;
```

- *ExpandUNCFileName(将文件名加在当前网络驱动器的UNC路径)*

引用单元：SysUtils

函数声明：function ExpandUNCFileName (const FileName: string): string;



C-16

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  FileName : String;
begin
  { 设置当前工作目录为网络邻居所连接而成的网络驱动器
    以 //carlos/temp 为例的 E 磁盘
    则其转换的名称为 //carlos/temp/MyFileName.bak }
  SetCurrentDir('E:\');
  FileName := 'MyFileName.bak';
  FileName := ExpandUNCFileName(FileName);
  showmessage(FileName);
end;
```

- *ExtractFileDrive(返回指定路径中的驱动器名称)*

引用单元：SysUtils
函数声明：function ExtractFileDrive(const FileName: string): string;



C-17

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(ExtractFileDrive('c:\temp\test.txt'));
end;
```

- *ExtractFileDir(返回指定路径中的目录名称)*

引用单元：SysUtils
函数声明：function ExtractFileDir(const FileName: string): string;



C-18

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(ExtractFileDir('c:\temp\test.txt'));
end;
```

- *ExtractFileExt(返回指定路径中的扩展名)*

引用单元：SysUtils
函数声明：function ExtractFileExt(const FileName: string): string;



C-19

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(ExtractFileExt('c:\temp\test.txt'));
end;
```

- *ExtractFileName(返回指定路径中的文件名)*

引用单元：SysUtils
函数声明：function ExtractFileName(const FileName: string): string;



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(ExtractFileName('c:\temp\test.txt'));
end;
```

- *ExtractFilePath(返回指定路径中的路径名称)*

引用单元：SysUtils
函数声明：function ExtractFileName(const FileName: string): string;



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(ExtractFilePath('c:\temp\test.txt'));
end;
```

- *ExtractRelativePath(返回具有关联性的路径名称)*

引用单元：SysUtils
函数声明：function ExtractRelativePath(const BaseName, DestName: string): string;



```
procedure TForm1.Button1Click(Sender: TObject);
var
   BaseName, DestName, RelativePath  : String;
begin
   BaseName := 'c:\temp\';
   DestName := 'c:\temp\test.txt';
   RelativePath := ExtractRelativePath(BaseName,DestName);
   ShowMessage(RelativePath);
end;
```

- *ExtractShortPathName(返回 DOS 模式下的 8.3 文件格式路径)*

引用单元：SysUtils
函数声明：function ExtractShortPathName(const FileName: string): string;



C-23

```
procedure TForm1.Button1Click(Sender: TObject);
var
   LongFileName, ShortFileName : String;
begin
   LongFileName := 'C:\Program Files\Borland\Delphi6\Bin
                                              \Delphi32.exe';
   ShortFileName := ExtractShortPathName(LongFileName);
   ShowMessage(ShortFileName);
end;
```

- *IncludeTrailingBackslash*(返回以“\”为结尾的路径名称)

引用单元：SysUtils

函数声明：function IncludeTrailingBackslash(const S: string): string;



C-24

```
procedure TForm1.Button1Click(Sender: TObject);
var
   oldPath, newPath : String;
begin
   oldPath := 'c:\temp';
   newPath := IncludeTrailingBackSlash(oldPath);
   ShowMessage(newPath);
end;
```

- *IncludeTrailingPathDelimiter*(返回以“\”为结尾的路径名称)

引用单元：SysUtils

函数声明：function IncludeTrailingPathDelimiter(const S: string): string;



C-25

```
procedure TForm1.Button1Click(Sender: TObject);
var
   oldPath, newPath : String;
begin
   oldPath := 'c:\temp';
   newPath := IncludeTrailingPathDelimiter(oldPath);
   ShowMessage(newPath);
end;
```

- *IsPathDelimiter(判断路径名称是否以“\”为结尾)*

引用单元：SysUtils
函数声明：function IsPathDelimiter(const S: string; Index: Integer): Boolean;

例 C-26

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Index : Integer;
   Path : String;
   ReturnValue : Boolean;
begin
   Path := 'c:\temp\';
   Index := Length(Path);
   ReturnValue := IsPathDelimiter(Path,Index);
   ShowMessage(BoolToStr(ReturnValue,True));
end;
```

- *MatchesMask(判断指定路径是否符合掩码条件)*

引用单元：Masks
函数声明：function MatchesMask(const Filename, Mask: string): Boolean;

例 C-27

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Filename, Mask : String;
   ReturnValue : Boolean;
begin
   //uses Masks
   Filename := 'Test.txt';
   Mask := 'T???.*';
   ReturnValue := MatchesMask(Filename,Mask);
   ShowMessage(BoolToStr(ReturnValue,True));
end;
```

C.3 文件属性处理函数

本小节将为您介绍 Delphi 6 所提供的文件属性处理函数，这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先行说明每一个目录处理功能函数所代表的意义，如表 C.3 所示。

表 C.3

函数名称	单元文件	所代表的意义
FileGetAttr	SysUtils	此函数可获取文件的属性，其返回值为整型类型。每一个整数值对应于 Delphi 6 所定义的常数值，包括：faReadOnly、faHidden、faSysFile、faVolumeID、faDirectory、faArchive 及 faAnyFile 等。当其返回值为-1 时，表示读取指定文件时发生错误
FileSetAttr	SysUtils	此函数可设置文件的属性值，其返回值为整型类型，当返回值为 0 代表更改成功，反之，则代表更改失败
FileAge	SysUtils	此函数可返回文件的日期时间，其时间格式为操作系统时间(OS timestamp)。您可以通过 FileDateToDateTime 函数将其转为 TDateTime 的日期格式。假如其返回值为-1 表示找不到指定的文件
FileIsReadOnly	SysUtils	此函数可返回指定文件对当前用户而言是否为只读权限。其返回值为布尔值，假如返回值为 True，表示只有读取的权限
FileDateToDateTime	SysUtils	此函数可以将 FileGetDate 或 FileAge 等函数所返回的操作系统时间(OS timestamp)格式转为 TDateTime 的日期格式
DateTimeToFileDate	SysUtils	此函数可以将 TDateTime 的日期格式转换为操作系统时间(OS timestamp)格式，以供 FileSetDate 函数设置文件的时间

以下将以范例的方式说明每一个函数的使用方法。

FileGetAttr(获取文件的属性)

引用单元：SysUtils

函数声明：function FileGetAttr(const FileName: string): Integer;

范例 C-28

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileName : String;
  Attrs : Integer;
begin
  FileName := 'C:\Temp\Backup\Test1.BAK';
  Attrs := FileGetAttr(FileName);
  if Attrs = -1 then Showmessage('查找文件发生错误')
  else begin
     if (Attrs and faReadOnly) <> 0 then
       Showmessage('此文件是只读文件')
     else
       Showmessage('此文件不是只读文件');
  end;
end;
```

FileGetAttr 函数的返回值都对应于 Delphi 所预先定义的常数，而且每一个文件可以同时拥有多个属性值。以下将以列表方式说明这些返回值为常数间的关系，如表 C.4 所示。

表 C.4

属性名称	常数值(十六进制)	所代表的意义
faReadOnly	$00000001	只读文件
faHidden	$00000002	隐藏文件
faSysFile	$00000004	系统文件
faVolumeID	$00000008	卷(Volume)类型
faDirectory	$00000010	目录
faArchive	00000020	存档文件
faAnyFile	$0000003F	正常文件

FileSetAttr(设置文件的属性)

引用单元：SysUtils

函数声明：function function FileSetAttr(const FileName: string; Attr: Integer): Integer;

范例 C-29

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileName : String;
  Attrs : Integer;
begin
  FileName := 'C:\Temp\Backup\Test1.BAK';
  Attrs := FileGetAttr(FileName);
  if Attrs = -1 then Showmessage('查找文件发生错误')
  else begin
     if (Attrs and faReadOnly) = 0 then begin
        Showmessage('此文件不是只读文件');
        //第一种设置方法(取消原有的属性
        //并同时设置只读、隐藏、存档等属性)
        //FileSetAttr(FileName,faReadOnly or faHidden or faArchive);
        //第二种设置方法(在原有的属性下加入只读属性)
        if FileSetAttr(FileName, Attrs + faReadOnly) = 0 then
           Showmessage('更改属性成功')
        else
           Showmessage('更改属性失败');
     end else
        Showmessage('此文件已经是只读文件');
  end;
end;
```

FileAge(返回文件的日期时间)

引用单元：SysUtils
函数声明：function FileAge(const FileName: string): Integer;

范例 C-30

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileName : String;
  Age : Integer;
  vDateTime : TDateTime;
begin
  FileName := 'c:\AUTOEXEC.BAT';
  Age := FileAge(FileName);
  if Age = -1 then
    Showmessage('找不到文件')
  else begin
    vDateTime := FileDateToDateTime(Age);
    showMessage(DateTimeToStr(vDateTime))
  end;
end;
```

FileIsReadOnly(返回文件是否为只读权限)

引用单元：SysUtils
函数声明：function FileIsReadOnly(const FileName: string): Boolean;

范例 C-31

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileName : String;
begin
  FileName := 'c:\AUTOEXEC.BAT';
  if FileIsReadOnly(FileName) then
    Showmessage('只读权限')
  else
    showMessage('非只读权限');
end;
```

C.4 文件处理函数

本小节将为您介绍 Delphi 6 所提供的文件处理函数，这些函数稍后将有详细的范例说

明。笔者这里将以列表的方式说明每一个文件处理函数所代表的意义，如表 C.5 所示。

表 C.5

函数名称	单元文件	所代表的意义
FileSearch	SysUtils	此函数可以在指定的路径下查找一个已知的文件名，其返回值为字符串类型。假如找到指定文件，此函数会返回该文件完整的路径。反之，则返回空字符串
FileExist	SysUtils	此函数可以判断指定的文件是否存在，您必须为这个文件指明完整的路径，否则系统则在当前的工作目录中查找。此函数的返回值为布尔值，当返回值为 True，表示找到指定的文件，反之，则代表找不到该文件
CopyFile	Windows API	CopyFile 函数是一个 Windows API，它可以进行单一文件的复制。此函数的返回值为布尔值，当返回值为 True，表示成功的复制文件，反之，则代表复制失败
MoveFile	Windows API	MoveFile 函数是一个 Windows API，它可以进行单一文件的移动，但其无法进行两个磁盘间的移动。如果您要进行两个磁盘间的文件移动，请先复制文件至目的磁盘，再删除原始的文件。此函数的返回值为布尔值，当返回值为 True，表示成功的移动文件，反之，则代表移动失败
MoveFileEx	Windows API	MoveFileEx 函数是一个 Windows API，它可以进行单一文件的移动，但其无法进行两个磁盘间的移动。此函数与 MoveFile 函数具有相同的功能，惟一的不同在于 MoveFileEx 函数可支持 Windows NT 的操作系统。此函数的返回值为布尔值，当返回值为 True，表示成功的移动文件，反之，则代表移动失败
DeleteFile	SysUtils	此函数可删除一个指定的文件，其返回值为布尔值。当文件被成功的删除，则其返回值为 True。反之，则代表指定的文件可能不存在或无法删除
RenameFile	SysUtils	此函数可更改一个指定文件的文件名，其返回值为布尔值。当文件名被成功的更改，则其返回值为 True。反之，则代表指定的文件可能不存在或新的文件名已经存在
FindFirst	SysUtils	此函数可依照指定的条件查找第一个符合条件的文件。其返回值为整型类型。当返回值为 0 代表已经找到文件，反之，则代表找不到指定文件。当文件被找到时，系统会将此文件的相关信息存储于第二个参数，其必须为 TSearchRec 类型
FindNext	SysUtils	此函数可依照指定条件查找下一个符合条件的文件，通常与 FindFirst 函数配合使用。其返回值与 FindFirst 函数相同
FindClose	SysUtils	结束文件查找函数，其目的是释放 FindFirst 或 FindNext 函数所占用的内存空间

以下将以范例的方式说明每一个函数的使用方法。

FileSearch(查找文件的正确路径)

引用单元：SysUtils
函数声明：function FileSearch(const Name, DirList: string): string;

范例 C-32

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Pi_eName, FilePath : String;
begin
  FileName := 'test1.bak';
  //此函数并不会搜索子目录
  FilePath :- FileSearch(FileName,'c:\temp\backup;c:\;d:\');
  if FilePath = '' then
     Showmessage('找不到文件')
  else
     ShowMessage(FilePath);
end;
```

FileExist(判断文件是否存在)

引用单元：SysUtils
函数声明：function FileExists(const FileName: string): Boolean;

范例 C-33

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileName : String;
begin
  FileName := 'c:\temp\backup\test1.bak';
  if FileExists(Filename) then
     Showmessage('找到文件')
  else
     Showmessage('找不到文件');
end;
```

CopyFile(复制文件)

引用单元：CopyFile 是一个 Windows API。
1. Windows API 的函数声明： //CopyFile 函数返回值为布尔值，假如成功则返回非零的数字，

```
    //反之返回零
    BOOL CopyFile(
      LPCTSTR lpExistingFileName,   //已经存在的文件名
      LPCTSTR lpNewFileName,        //被复制的文件名
      BOOL bFailIfExists            //文件已经存在时的处理方式
                                    //False表示覆盖，True表示中止
    );
```

2. Delphi 的函数用法：

```
function CopyFile(SourceFileName : PChar;
  Destination FileName : PChar;Flag : Boolean): Boolean;
```

3. 实例说明：

以下程序片段，说明如何将C:\Autoexec.bat复制为C:\test.txt

```
CopyFile(pchar('c:\autoexec.bat'),pchar('c:\test.txt'),false);
```

MoveFile(移动文件)

引用单元：MoveFile是一个Windows API。

1. Windows API 的函数声明：

```
    //MoveFile函数返回值为布尔值，假如成功则返回非零的数字，
    //反之返回零
    BOOL CopyFile(
      LPCTSTR lpExistingFileName,  //已经存在的文件名
      LPCTSTR lpNewFileName,       //被移动之后的文件名
    );
```

2. Delphi 的函数用法：

```
function MoveFile(SourceFileName : PChar;
    Destination FileName : PChar): Boolean;
```

3. 实例说明：

以下程序片段，说明如何将C:\test.txt 移动至C:\temp\test2.txt

```
MoveFile(pchar('c:\test.txt'),pchar('c:\temp\test2.txt'));
```

MoveFileEx(移动文件，Windows NT专用)

引用单元：MoveFileEx是一个Windows API。

4. Windows API 的函数声明：

```
    //MoveFile函数返回值为布尔值，假如成功则返回非零的数字，
    //反之返回零
    BOOL CopyFile(
      LPCTSTR lpExistingFileName,    //已经存在的文件名
      LPCTSTR lpNewFileName,         //被移动之后的文件名
      DWORD dwFlags                  //文件存在时的处理方式
    );
```

5. Delphi 的函数用法：

```
function MoveFileEx(SourceFileName : PChar;
                    Destination FileName : PChar
                    Flag : Word): Boolean;
```

6. 实例说明：

以下程序片段，说明如何将 C:\test.txt 移动至 C:\temp\test2.txt，假如 test2.txt 文件已经存在，则覆盖

```
MoveFileEx(pchar('c:\test.txt'),pchar('c:\temp\test2.txt'),
                MOVEFILE_REPLACE_EXISTING);
```

DeleteFile(删除文件)

引用单元：SysUtils

函数声明：function DeleteFile(onst FileName: string): Boolean;

范例 C-34

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if FileExists('c:\autoexec.bak')then
   if DeleteFile('c:\autoexec.bak') then
      ShowMessage('删除成功')
   else
      ShowMessage('删除失败')
else
   ShowMessage('指定文件不存在');
end;
```

RenameFile(更改文件名)

引用单元：SysUtils

函数声明：function RenameFile(const OldName, NewName: string): Boolean;

范例 C-35

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if FileExists('c:\autoexec.bak')then
   if RenameFile('c:\autoexec.bak','c:\autoexec.kkk') then
      ShowMessage('更名成功')
   else
      ShowMessage('更名失败')
else
   ShowMessage('指定文件不存在');
end;
```

- *FindFirst(查找第一个符合条件的文件)*

引用单元：SysUtils
函数声明：function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;

- *FindNext(查找下一个符合条件的文件)*

引用单元：SysUtils
函数声明：function FindNext(var F: TSearchRec): Integer;

- *FindClose(结束文件查找)*

引用单元：SysUtils
函数声明：procedure FindClose(var F: TSearchRec);

FindFirst、FindNext 与 FindClose 函数有一个 TSearchRec 类型的共同的参数，其代表被找到的文件相关信息。TSearchRec 是一个记录数据类型，其原始声明如下：

```
type
TSearchRec = record
                Time: Integer;
                Size: Integer;
                Attr: Integer;
                Name: TFileName;
                ExcludeAttr: Integer;
                FindHandle: THandle;
                FindData: TWin32FindData;
end;
```

其相关的字段说明如下：

- Time 代表文件相关的时间，您可以通过 FileDateToDateTime 函数将其转为 TDateTime 的日期时间格式。
- Size 代表文件大小其计算单位为字节(bytes)。
- Attr 代表文件的相关属性值，有关文件属性的详细介绍请参考 C.3 小节。
- Name 代表文件名，其包含扩展名。
- ExcludeAttr 代表其他不包含的文件属性。
- FindHandle 代表内部的文件句柄(File Handle)。
- FindData 代表此这文件其他的相关信息，例如：文件的创建时间、文件最后的存取时间、长的文件名和短的文件名等。



C-36

目的：本范例将以递归的方式找出指定目录下的所有文件。

1 创建新的工程。

(2) 在 TForm 对象上加入一个 TMemo 对象、一个 TEdit 对象、一个 TLabel 对象及一个 TButton 对象，其相关属性设置如表 C.6 所示。

表 C.6

对象名称	属性	属性值
TMemo	Name	Memo1
TEdit	Name	Edit1
TLabel	Name	Label1
	Caption	目录名称:
TButton	Name	Button1
	Caption	Get All Files

(3) 声明一个用户自定义函数，其名称为 GetAllFiles。

```
  public
    { Public declarations }
  procedure GetAllFiles(SourcePath:String);
```

(4) 为自定义函数 GetAllFiles 加入以下的代码。此段代码的目的是以递归的方式获取指定目录下所有的文件名。

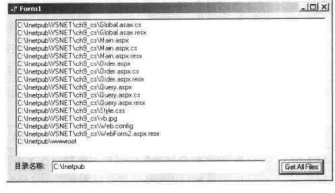
```
procedure TForm1.GetAllFiles(SourcePath:String);
var
  sr: TSearchRec;
begin
  //为指定的目录名称加上 "\" 符号。
  SourcePath := IncludeTrailingBackslash(SourcePath);
  //查找第一个符合条件的文件
  if FindFirst(SourcePath + '\*.*' , faAnyFile, sr) = 0 then begin
     repeat
       if (sr.Name <> '.') and (sr.Name <> '..') then begin
         if sr.Attr <> faDirectory then begin
           Memo1.Lines.Append(SourcePath + sr.Name);
         end else begin
           //如果是子目录，则以递归的方式调用此函数
           //，再找出其他的文件
           GetAllFiles(SourcePath + sr.Name)
         end;
       end;
     //继续查找下一个符合条件的文件
     until FindNext(sr) <> 0;
  end;
  //结束搜索，以释放 sr 所占用的内存
  FindClose(sr);
end;
```

步骤5 为TButton对象的onClick事件处理过程中加入以下的代码。此段代码的目的是调用GetAllFiles函数，并将其搜索结果显示于TMemo对象。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   GetAllFiles(Edit1.text);
end;
```

步骤6 保存工程。

步骤7 按【F9】功能键运行工程后，先输入特定目录后再按下TButton对象按钮。此时系统将会显示指定目录下的所有文件，这里以C:\Inetpub为例，如图C.1所示。



图C.1 FindFirst、FindNext与FindClose函数应用范例

C.5 文本文件处理函数

实际上，一个企业不可能只存在一种应用系统，而且不同应用系统通常也不是独立的。换言之，在众多的系统间经常需要做跨系统的数据交换。面对不同的操作系统与不同的应用程序，文本文件是一个不错的数据交换介质。本小节将为您介绍Delphi 6所提供的文本文件处理函数，这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先行说明每一个文本文件处理函数所代表的意义，如表C.7所示。

表C.7

函数名称	单元文件	所代表的意义
AssignFile	System	此函数可以通过文件变量连接一个外部文件。您不可以使用AssignFile函数连接至一个已经打开的文件
CloseFile	System	CloseFile函数将会关闭一个由AssignFile函数所打开的文件
Reset	System	Rewrite函数将以只读的方式打开指定文件，通常配合AssignFile函数一起使用
Eof	System	EOF函数可判断当前文件指针是否已经到文件结尾
Readln	System	Readln函数会以行为单位读取文本文件的内容
Eoln	System	Eoln函数可判断当前文件指针是否已经到一行的结尾

续上表

函数名称	单元文件	所代表的意义
Read	System	Read 函数会以字为单位读取文本文件的内容
Append	System	Append 函数可以打开一个可附加的文件，通常必须配合 AssignFile 函数一起使用。Append 函数与 Rewrite 函数最大的不同在于后者会清除已经存在的数据，而前者则会保留原有的数据。另外一个非常重要的不同点在于 Append 函数与 Reset 函数所对应的文件必须已经存在。而 Rewrite 函数若对应一个不存在的文件时，则会自行创建该文件
Flush	System	当您调用 Flush 函数时，系统会将尚未写入文件的暂存数据，加以更新到文件中。如果您的程序要将很重要的数据写入文件中，而且文件可能会打开很久的时间。笔者建议您在调用 CloseFile 函数之前，可以先使用 Flush 函数将数据写入文件，以确保数据的正确性
Rewrite	System	Rewrite 函数将以可读可写的模式打开指定文件，通常必须配合 AssignFile 函数一起使用
Write	System	Write 函数可以将指定变量的内容写入文本文件中
Writeln	System	Writeln 函数可将数据以行为单位写入文本文件
SeekEoln	System	Eoln 函数可以判断当前文件指针是否已经到一行的结尾，这里的判断条件包含所有的字符，无论是空格符或是每一行的结束符号。而 SeekEoln 函数只针对非空格符的部分做判断
SeekEof	System	SeekEof 函数与 SeekEoln 函数一样会含去空格符，但前者是针对整个文件，而后者是针对文件中的某一行
SetLineBreakStyle	System	SetLineBreakStyle 函数可决定文本文件每一行的结束符号，其包括 tlbsLF 与 tlbsCRLF 两种
Truncate	System	Truncate 函数可以删除文件指针之后的文件内容

以下将以范例的方式说明每一个函数的使用方法。

- *AssignFile(通过文件变量连接一个外部文件)*

引用单元：System
函数声明：procedure AssignFile(var F; FileName: string);

- *CloseFile(关闭使用 AssignFile 函数所打开的文件)*

引用单元：System
函数声明：procedure CloseFile(var F);

- *ReSet(打开以只读的模式打开指定文件)*

引用单元：System
函数声明：procedure Reset(var F [: File; RecSize: Word]);

- *EOF(判断文件指针是否已经到文件结尾)*

引用单元：SysUtils
函数声明：function Eof(var F): Boolean; function Eof [(var F: Text)]: Boolean;

- *ReadLn(以行为单位读取文本文件内容)*

引用单元：System
函数声明：procedure Readln([var F: Text;] V1 [, V2, ...,Vn]);



C-37

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F1: TextFile;
  FileName, TmpStr, AllStr : String;
begin
  FileName := 'C:\Temp\test.txt';
  //打开指定的文件
  AssignFile(F1,FileName);
  Reset(F1);
  AllStr := '';
  //利用循环将 F1 的文件内容显示
  while not Eof(F1) do begin
    Readln(F1, TmpStr);
    //为每一行之间加入换行符号
    AllStr := AllStr + TmpStr + #10 ;
  end;
  //关闭文件
  CloseFile(F1);
  //显示文件的所有内容
  ShowMessage(AllStr);
end;
```

- *Eoln(判断文件指针是否已经到一行的结尾)*

引用单元：SysUtils
函数声明：function Eoln [(var F: Text)]: Boolean;

- *Read(以字为单位读取文件)*

引用单元：System
函数声明：procedure Read(F , V1 [, V2,...,Vn]); procedure Read([var F: Text;] V1 [, V2,...,Vn]);

范例 C-38

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F1: TextFile;
  FileName, AllStr : String;
  TmpStr : Char; //请注意 Readln 与 Read 两个函数范例的不同之处
begin
  FileName := 'C:\Temp\test.txt';
  //打开指定的文件
  AssignFile(F1,Filename);
  Reset(F1);
  AllStr := '';
  //利用循环将 F1 的文件内容显示
  while not Eof(F1) do begin
    Read(F1, TmpStr);
    //请特别注意，您不需要加上换行符号，
    //因为 Read 函数也会读取每一行的结尾符号
    AllStr := AllStr + TmpStr;
  end;
  //关闭文件
  CloseFile(F1);
  //显示文件的所有内容
  ShowMessage(AllStr);
end;
```

Append(打开一个可附加的文件)

引用单元：System

函数声明：procedure Append(var F: Text);

范例 C-39

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Filename : String;
   F: TextFile;
begin
   FileName := 'C:\Temp\Test.txt';
   //指定文件必须存在
   AssignFile(F,FileName);
   Append(F);
   Writeln(F, '附加的时间:' + DateTimeToStr(Now));
```

```
    CloseFile(F);
    ShowMessage('数据附加完成');
end;
```

- *Flush(更新文件内容)*

引用单元：System
函数声明：procedure Flush(var F: Text);

例 C-40

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Filename : String;
    F: TextFile;
begin
    FileName := 'C:\Temp\Test.txt';
    //指定文件必须存在
    AssignFile(F,FileName);
    Append(F);
    Writeln(F, '附加的时间:' + DateTimeToStr(Now));
    Showmessage('如果这个时候将程序中断，数据尚未写入');
    Flush(F); //将暂存数据写回文件
    Showmessage('数据已经成功写入，程序中断不会有影响');
    CloseFile(F);
    ShowMessage('数据附加完成');
end;
```

- *ReWrite(以可读可写模式打开指定文件)*

引用单元：System
函数声明：procedure Rewrite(var F: File [; Recsize: Word]);

- *Write(将指定变量的内容写入文本文件中)*

引用单元：System
函数声明：procedure Write([var F: Text;] P1 [, P2,..., Pn]);

例 C-41

```
procedure TForm1.Button1Click(Sender: TObject);
var
  oldFilename, newFileName : String;
  F1, F2: TextFile;
  Ch: Char;
  Count : Integer;
```

```
begin
  Count := 0; //计数器
  oldFileName := 'C:\Temp\Test.txt';
  newFileName := 'C:\Temp\Test.bak';
  //打开被拷贝的文件
  AssignFile(F1,oldFileName);
  Reset(F1);
  //打开要存放的文件(会被覆盖)
  AssignFile(F2,newFileName);
  Rewrite(F2);
  //利用循环将 F1 的文件内容拷贝到 F2
  while not Eof(F1) do
  begin
    //一次只读一个字符
    Read(F1, Ch);
    Write(F2, Ch);
    Count := Count + 1;
  end;
  //关闭文件
  CloseFile(F2);
  CloseFile(F1);
  //请比较 Write 与 Writeln 函数的不同之处
ShowMessage('复制完成, 读取与写入动作共'+IntToStr(Count)+'次');
end;
```

WriteLn(以行为单位将数据写入文本文件)

引用单元：System
函数声明：procedure Writeln([var F: Text;] P1 [, P2, ...,Pn]);

范例 C-42

```
procedure TForm1.Button1Click(Sender: TObject);
var
  oldFilename, newFileName : String;
  F1, F2: TextFile;
  Str: String;
  Count : Integer;
begin
  Count := 0; //计数器
  oldFileName := 'C:\Temp\Test.txt';
  newFileName := 'C:\Temp\Test.bak';
```

```
  //打开被拷贝的文件
  AssignFile(F1,oldFileName);
  Reset(F1);
  //打开要存放的文件(会被覆盖)
  AssignFile(F2,newFileName);
  Rewrite(F2);
  //利用循环将 F1 的文件内容拷贝到 F2
  while not Eof(F1) do
  begin
     //一次只读一行
     Readln(F1, Str);
     Writeln(F2, Str);
     Count := Count + 1;
  end;
  //关闭文件
  CloseFile(F2);
  CloseFile(F1);
  //请比较 Write 与 Writeln 函数的不同之处
ShowMessage('复制完成, 读取与写入操作共'+IntToStr(Count)+'次');
end;
```

SeekEoln(判断文件中某一行是否结束，其只针对非空格符)

引用单元：System
函数声明：function SeekEoln [(var F: Text)]: Boolean;

例 C-43

```
procedure TForm1.Button1Click(Sender: TObject);
var
   F: TextFile;
   i: Integer;
   c : Char;
   str : string;
begin
   AssignFile(F,'Test.TXT');
   Rewrite(f);
   For i := 0 to 9 do begin
      Write(F,i);
      Write(F,'    '); //写入 4 个空格符
   end;
   Writeln(F); //写入该行的结束符号
   Reset(F); //将数据写回文件，并且以只读的方式打开
```

```
  Str := '';
  while not SeekEoln(f) do //舍去空格符
  begin
    Read(F, C);
    Str := Str + C;
  end;
  Showmessage(Str); //显示第一行的结果
  CloseFile(F); //关闭文件
end;
```

SeekEOf(判断文本文件是否结束，其不包含空白行)

引用单元：System

函数声明：function SeekEof [(var F: Text)]: Boolean;

C-44

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F: TextFile;
  i: Integer;
  Str : string;
begin
  AssignFile(F,'Test.TXT');
  Rewrite(f);
  For i := 0 to 9 do begin
     if i mod 2 = 1 then Writeln(F,i) //写入变量内容
     else Writeln(F); //写入空白行
  end;
  //一共写入 10 行
  Reset(F); //将数据写回文件，并且以只读的方式打开
  i := 0;
  while not SeekEof(F) do //舍去空白行
  begin
    Readln(F,Str);
    i := i + 1;
  end;
  CloseFile(F); //关闭文件
  //一共有 5 行的非空白行
  Showmessage('共有' + IntToStr(i) + '行的非空白行');
end;
```

Truncate(删除文件指针之后的文件内容)

引用单元：System
函数声明：procedure Truncate(var F);

例 C-45

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file of Byte;
  size : Longint;
begin
    AssignFile(f,'c:\temp\test.txt');
    Reset(f);
    //获取删除前的文件大小
    size := FileSize(F);
    Showmessage('文件大小为：' + IntToStr(size));
    //将文件指针移至指定位置，并加以删除
    Seek(f,size div 2);
    Truncate(F);
    //获取删除后的文件大小
    size := FileSize(F);
    Showmessage('删除后的文件大小为：' + IntToStr(size));
    CloseFile(f);
end;
```

C.6 类型文件(Typed File)处理函数

在上一小节中，我们介绍了 Delphi 6 所提供的标准文本文件处理函数，因此每一个范例所声明的文件变量都必须使用 TextFile 数据类型。事实上，您也可以自行定义文件的类型、所谓文件，是指具有相同数据类型的元素所组合而成的一个有序集合。因此您可以舍弃 TextFile 或 Text 等预设的数据类型，而使用自行定义的文件类型。您可以使用下列的语法来声明文件类型：

```
Type
  YourFileTypename = file of YourFileType
```

其中 YourFileTypeName 可以是任何合乎语法规则的标识符，而 YourFileType 则是一个固定大小的数据类型。因为必须是固定大小，所以指针类型、动态数组、类别、对象及可变量据类型等不可以使用。您必须特别注意，如果使用 file of 来声明文件变量，那么它就是一个类型文件(Typed File)。相对的，如果使用 file 保留字来声明文件变量，那么它则是一个无类

型文件(Untyped File)。有关这个部分，笔者下一小节中将有详细的介绍说明。

Delphi 6 所提供的类型文件(Typed File)处理函数有很多都与章节 C.5 所介绍的一样。笔者这里就不再赘述，以下仅就其特有的函数加以说明，如表 C.8 所示。

表 C.8

函数名称	单元文件	所代表的意义
FileSize	System	FileSize 函数可以获取当前文件中的数据条数。其计算单位为您自行定义的文件类型大小
FilePos	System	FilePos 函数可以获取当前文件中的指针位置。其计算单位为您自行定义的文件类型大小
Seek	System	Seek 函数可以按照您自行定义的文件类型大小移动文件指针
Read	System	Read 函数可以依照您自行定义的文件类型大小从文件中读取数据
Write	System	Write 函数可以将指定变量的内容写入文本文件中。这里的变量必须是您自行定义的文件类型

接着笔者将以一个学生基本数据建档操作来说明如何使用具有类型(Typed File)的文件来保存数据。

范例 C-46

目的：本范例将通过学生基本数据的建档操作来说明如何使用有类型的文件。

STEP 1 创建新的工程。

STEP 2 在 TForm 对象上加入五个 TLabeledEdit 对象、四个 TLabel 对象及五个 TBitBtn 对象，其相关属性设置如表 C.9 所示。

表 C.9

对象名称	属性	属性值
TLabeledEdit	Name	EditID
	EditLabel.Caption	ID
TLabeledEdit	Name	EditName
	EditLabel.Caption	Name
TLabeledEdit	Name	EditAge
	EditLabel.Caption	Age
TLabeledEdit	Name	EditSex
	EditLabel.Caption	Sex
TLabeledEdit	Name	EditAddress
	EditLabel.Caption	Address
TLabel	Name	LabelTotal
	Caption	当前总条数：

续上表

对象名称	属性	属性值
TLabel	Name	LabelTotalRecords
	Caption	(空白)
TLabel	Name	LabelCurrent
	Caption	当前第几条:
TLabel	Name	LabelCurrentRecord
	Caption	(空白)
TBitBtn	Name	BitBtnFirst
	Caption	第一条
TBitBtn	Name	BitBtnPrior
	Caption	上一条
TBitBtn	Name	BitBtnNext
	Caption	下一笔
TBitBtn	Name	BitBtnLast
	Caption	最后一条
TBitBtn	Name	BitBtnSave
	Caption	添加

步骤 3 定义一个记录类型做为每一个学生记录的存取单位，其名称为 StudentRecord。

```
type
  //定义一个记录(Record)类型
  StudentRecord = Record
    ID : String[10];
    Name : String[10];
    Age : String[3];
    Sex : String[2]; //男、女
    Address : String[100];
  end;
```

步骤 4 声明所有必要的全局变量，包括常数与变量。其中 RecordSize 代表每一笔学生数据长度的常数、StudentData 代表记录学生记录的变量、StudentFile 代表以学生记录为主的文件变量。

```
const
  //计算出每一个学生数据的长度
  RecordSize = SizeOf(StudentRecord);

var
  Form1: TForm1;
  //定义学生记录变量
  StudentData : StudentRecord;
```

```
//定义文件变量
StudentFile : File of StudentRecord;
```

STEP 5 声明两个用户自定义函数，其名称为 DisplayRecord 与 DisplayCountInfo。前者负责显示当前文件指针所对应的学生记录，后者负责显示当前的数据总条数及指针位置。

```
public
  { Public declarations }
  //显示学生数据
  procedure DisplayRecord;
  procedure DisplayCountInfo;
```

STEP 6 为自定义函数 DisplayRecord 加入以下的代码。此段代码的目的是显示当前文件指针所对应的学生记录。

```
procedure TForm1.DisplayRecord;
begin
  //读取数据
  Read(StudentFile,StudentData);
  EditID.Text := StudentData.ID;
  EditName.Text := StudentData.Name;
  EditAge.Text := StudentData.Age;
  EditSex.Text := StudentData.Sex;
  EditAddress.Text := StudentData.Address;
  //显示当前的条数信息
  DisplayCountInfo;
  //判断哪些按钮可以有作用
  if FileSize(StudentFile) > 0 then begin
    BitBtnFirst.Enabled := FilePos(StudentFile) <> 1;
    BitBtnLast.Enabled := FilePos(StudentFile) <>
                                        FileSize(StudentFile);
    BitBtnNext.Enabled := FilePos(StudentFile) <
                                        FileSize(StudentFile);
    BitBtnPrior.Enabled := FilePos(StudentFile) > 1;
  end else begin
    BitBtnFirst.Enabled := False;
    BitBtnLast.Enabled := False;
    BitBtnNext.Enabled := False;
    BitBtnPrior.Enabled := False;
  end;
end;
```

STEP 7 为自定义函数 DisplayCountInfo 加入以下的代码。此段代码的目的是显示当前数据总条数及指针位置。

```
procedure TForm1.DisplayCountInfo;
```

```
begin
  LabelCurrentRecord.Caption := IntToStr(FilePos(StudentFile));
  LabelTotalRecords.Caption := IntToStr(FileSize(StudentFile));
end;
```

STEP 8 为 TForm 对象的 onCreate 事件处理过程加入以下的代码，这段代码主要目的是打开指定文本文件并显示第一条记录内容。

```
procedure TForm1.FormCreate(Sender: TObject);
var
  FileName : String;
begin
  FileName := 'Students.txt';
  //打开学生数据文件
  AssignFile(StudentFile, FileName);
  //根据文件是否存在，决定使用 Reset 或 Rewrite
  if FileExists(FileName) then Reset(StudentFile)
  else Rewrite(StudentFile);
  //如果文件有数据，则加以显示
  if NOT EOF(StudentFile) then DisplayRecord
  //显示当前的条数信息
  else DisplayCountInfo;
end;
```

STEP 9 为 TForm 对象的 onClose 事件处理过程加入以下的代码，这段代码主要目的是关闭指定的文本文件。

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  //关闭文件
  CloseFile(StudentFile);
end;
```

STEP 10 为 BitBtnFirst 对象的 onClick 事件处理过程加入以下的代码，这段代码主要目的是显示第一条的记录内容。

```
procedure TForm1.BitBtnFirstClick(Sender: TObject);
begin
  //第一条
  Seek(StudentFile,0);
  //显示数据
  DisplayRecord;
end;
```

STEP 11 为 BitBtnPrior 对象的 onClick 事件处理过程加入以下的代码，这段代码主要目的是将文件指针往前移动一条记录并显示其内容。

```
procedure TForm1.BitBtnPriorClick(Sender: TObject);
begin
  //移动指针
  Seek(StudentFile,FilePos(StudentFile)-2);
  //显示数据
  DisplayRecord;
end;
```

(12) 为 BitBtnNext 对象的 onClick 事件处理过程加入以下的代码，这段代码主要目的是将文件指针往下移动一笔记录并显示其内容。

```
procedure TForm1.BitBtnNextClick(Sender: TObject);
begin
  //移动指针
  Seek(StudentFile,FilePos(StudentFile));
  //显示数据
  DisplayRecord;
end;
```

(13) 为 BitBtnLast 对象的 onClick 事件处理过程加入以下代码，这段代码主要目的是将文件指针移至最后一条记录并显示其内容。

```
procedure TForm1.BitBtnLastClick(Sender: TObject);
begin
  //最后一条
  Seek(StudentFile,FileSize(StudentFile)-1);
  //显示数据
  DisplayRecord;
end;
```

(14) 为 BitBtnSave 对象的 onClick 事件处理过程加入以下的代码，这段代码主要目的是创建一笔新的记录。笔者这里将创建新记录的动作分为两个小步骤，第一个步骤只是清除画面并将按钮标题文字改为“保存”，第一个步骤才是真正将数据保存到文件。

```
procedure TForm1.BitBtnSaveClick(Sender: TObject);
begin
  if BitBtnSave.Caption = '保存' then begin
    //移动指针
    Seek(StudentFile,FileSize(StudentFile));
    //将数据写入当前指针位置
    StudentData.ID := EditID.Text;
    StudentData.Name := EditName.Text;
    StudentData.Age := EditAge.Text;
    StudentData.Sex := EditSex.Text;
    StudentData.Address := EditAddress.Text;
    write(StudentFile,StudentData);
```

```
      BitBtnSave.Caption := '添加';
      BitBtnSave.Enabled := True;
    end else begin
      BitBtnSave.Caption := '保存';
      EditID.Text := '';
      EditName.Text := '';
      EditAge.Text := '';
      EditSex.Text := '';
      EditAddress.Text := '';
      BitBtnSave.Enabled := False;
    end;
    //显示当前的条数信息
    DisplayCountInfo;
  end;
```

15 为了数据的有效性，请在 EditID 对象的 onChange 事件处理过程加入以下的代码，并且将其与 EditName、EditAge、EditSex 及 EditAddress 等对象共同此事件处理过程。这段代码主要的目的中决定“保存”按钮的有效性。

```
procedure TForm1.EditIDChange(Sender: TObject);
begin
  //设置保存的条件
  BitBtnSave.Enabled := (EditID.Text <> '') and
    (EditName.Text <> '') and (EditAge.Text <> '') and
    (EditSex.Text <> '') and  (EditAddress.Text <> '');
end;
```

16 保存并运行工程。工程运行后，您可以创建新的记录或浏览已保存的学生基本数据，如图 C.2 所示。



图 C.2 类型文件(Typed File)处理函数应用范例

C.7 无类型文件(Untyped File)处理函数

到当前为止，我们已经介绍了标准文本文件与类型文件(Typed File)的相关处理函数。本小节将继续为您介绍另一种文件处理函数，我们称它为无类型文件(Untyped File)处理函数。所谓无类型文件，也就是说 Delphi 并没有提供对应的数据类型，例如可执行文件、图形文件、

图像文件、声音文件或是其他应用程序所指定的文件，如 MP3、.ppt、.doc 等，都可以称为“无类型文件”。

在前面的章节中笔者曾经做了一个简单的比较，也就是说，如果使用 TextFile 或 Text 等数据类型来声明文件变量，那么它就是所谓的标准文本文件；如果使用 file of 保留字来声明文件变量，那么它就是所谓的类型文件；如果使用 file 保留字来声明文件变量，那么它就是一个无类型文件。Delphi 6 所提供的无类型文件(Untyped File)处理函数有很多都与章节 C.5 所介绍的相同。笔者这里就不再赘述，以下仅就其特有的函数加以说明，如表 C.10 所示。

表 C.10

函数名称	单元文件	所代表的意义
BlockRead	System	BlockRead 函数可从指定的外部文件中读取一个或一个以上的记录。
BlockWrite	System	BlockWrite 函数可将一个或一个以上的记录写到指定的外部文件中。

BlockRead 函数的原始声明如下：

```
procedure BlockRead(var F: File; var Buf; Count: Integer
                    [; var AmtTransferred: Integer]);
```

BlockRead 函数有 F、Buf、Count 与 AmtTransferred 等四个参数，分述如下：

- F 参数代表指定的文件变量，它必须是一个由 File 保留字所声明的文件变量。
- Buf 参数代表一份内存空间负责保存由文件读取的数据，它可以是任何类型的变量。
- Count 参数则代表 Buf 参数的内存空间大小，它是一个整型类型的变量。通常我们会使用 SizeOf 函数来获取 Buf 参数所占的内存大小。BlockRead 函数每一次从文件读取的数据量会小于或等于 Count 参数。
- AmtTransferred 参数代表 BlockRead 函数已经读入的数据量，当指定的数据量(Count)已完全读入时，AmtTransferred 参数会等于 Count 参数。当这两个参数不相等时，代表已经读取到文件结尾或是发生读取上的错误。您可以省略此参数。

BlockWrite 函数的原始声明如下：

```
procedure BlockWrite(var F: File; var Buf; Count: Integer
                     [; var AmtTransferred: Integer]);
```

BlockWrite 函数也有 F、Buf、Count 与 AmtTransferred 等四个参数。这些参数的用法与 BlockRead 函数相同，其间的差异在于一个是写入而另一个是读取而已。这些参数的用法请参考 BlockRead 函数的详细说明。

BlockRead 与 BlockWrite 两个函数在处理无类型文件时最常被使用的函数。当然您必须配合 AssignFile 函数打开文件、Reset 或 Rewrite 函数决定打开的模式，最后您也必须使用 CloseFile 函数来关闭已经打开的文件以利内存的释放。接着笔者将以一个文件复制的范例来说明如何使用 Delphi 6 所提供的无类型的文件处理函数。

范例 C-47

目的：本范例修改自范例 C-36，将原本以递归方式找出指定目录下所有文件名的功能加入文件复制功能。

Step 1 打开范例 C-36。

Step 2 在 TForm 对象上另外加入一个 TEdit 对象及一个 TLabel 对象，其相关属性设置如表 C.11 所示。

表 C.11

对象名称	属性	属性值
TEdit	Name	Edit2
TLabel	Name	Label2
	Caption	目标目录:

Step 3 声明一个用户自定义函数，其名称为 myCopyFile。此函数的两个参数分别代表源文件与目标文件的完整路径名称。

```
public
  { Public declarations }
  procedure myCopyFile(SourcePath,TargetPath:String);
```

Step 4 为自定义函数 myCopyFile 加入以下的代码。此段代码的目的主要是通过 Delphi 6 所提供的无类型文件函数进行文件复制。

```
procedure TForm1.myCopyFile(SourcePath,TargetPath:String);
var
  SourceFile,TargetFile : File;
  SourceCount, TargetCount : Integer;
  Buffer: array[1..2048] of Char; //大小可自行决定
begin
  //假如目标目录不存在，先行创建
  ForceDirectories(ExtractFileDir(TargetPath));
  //指定源文件与目的文件
  AssignFile(SourceFile, SourcePath);
  Reset(SourceFile, 1); //读取的数据条数为 1
  AssignFile(TargetFile, TargetPath);
  Rewrite(TargetFile, 1); //写入的数据条数为 1
  repeat
    //读取指定 Buffer 大小的数据
    BlockRead(SourceFile, Buffer, SizeOf(Buffer), SourceCount);
    //写入指定 Buffer 大小的数据
    BlockWrite(TargetFile, Buffer, SourceCount, TargetCount);
  //判断结束的条件
  until (SourceCount = 0) or (TargetCount <> SourceCount);
```

```
  //关闭文件
  CloseFile(SourceFile);
  CloseFile(TargetFile);
end;
```

步骤 5 改写 TButton 对象的 onClick 事件处理过程。此段代码的目的是调用 GetAllFiles 函数并进行文件复制的工作，其中第一个参数代表来源目录，第二个参数代表目标目录。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   GetAllFiles(Edit1.text,Edit2.Text);
end;
```

步骤 6 因为 GetAllFiles 函数必须使用“源目录”与“目标目录”两个参数，因此我们必须修改其函数的声明。

```
procedure GetAllFiles(SourcePath,TargetPath:String);
```

步骤 7 GetAllFiles 函数除了声明部分需要修改外，其函数的内容也要做适当的调整。

```
procedure TForm1.GetAllFiles(SourcePath,TargetPath:String);
var
  sr: TSearchRec;
begin
  //为指定的目录名称加上“\”符号。
  SourcePath := IncludeTrailingBackslash(SourcePath);
  TargetPath := IncludeTrailingBackslash(TargetPath);
  //查找第一个符合条件的文件
  if FindFirst(SourcePath + '\*.*' , faAnyFile, sr) = 0 then begin
    repeat
      if (sr.Name <> '.') and (sr.Name <> '..') then begin
        if sr.Attr <> faDirectory  then begin
          Memo1.Lines.Append(SourcePath + sr.Name);
          //调用复制的子程序，并且传入目的目录
          MyCopyFile(SourcePath + sr.Name,TargetPath +
                                                    sr.Name);
        end else begin
          //如果是子目录，则以递归的方式调用此函数
          //，再找出其他的文件
          GetAllFiles(SourcePath + sr.Name,TargetPath +
                                                    sr.Name)
        end;
      end;
    //继续查找下一个符合条件的文件
    until FindNext(sr) <> 0;
  end;
```

```
    //结束搜索，以释放 sr 所占用的内存
    FindClose(sr);
  end;
```

8 保存工程。

9 按【F9】功能键运行工程后，先输入“源目录”与“目标目录”后再按下“TButton”对象按钮。此时系统将会显示且进行文件的复制工作，这里以 C:\Inetpub 为例，如图 C.3 所示。

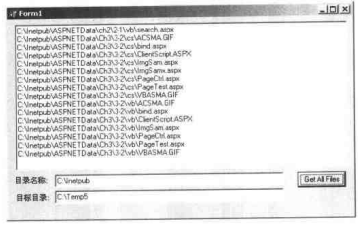


图 C.3 无类型文件处理函数应用范例

附 录 D

Delphi 6 常用日期与时间函数

本 章 目 标

日期与时间对程序设计师而言是一个非常重要的数据类型。因为其可能随着不同的区域设置而有不同的日期与时间格式，例如长的日期格式或短的日期格式等。除了操作系统的时间外，还有一个日期时间格式则来自数据库。编写数据库应用程序的程序设计师都知道，每一台客户端计算机的时日期间可能不尽相同，因此必须依赖数据库所给予的标准时间来进行判断。希望通过完整的函数介绍可协助您正确与快速地处理这些问题。

本 章 主 题

D.1 获取特定的日期与时间

D.2 日期处理函数

D.3 时间处理函数

D.4 比较与判断函数

D.5 日期与时间的编译与解译

D.6 日期与时间的转换函数

D.1 获取特定的日期与时间

本小节将为您介绍 Delphi 6 所提供获取特定日期与时间的函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 D.1 所示。

表 D.1

函数名称	单元文件	所代表的意义
Now	SysUtils	此函数可返回现在的日期与时间，其返回值为 TDateTime 类型
Date	SysUtils	此函数可返回现在的日期，其返回值为 TDateTime 类型
Time	SysUtils	此函数可返回现在的时间，其返回值为 TDateTime 类型
Today	DateUtils	此函数可返回今天的日期，其返回值为 TDateTime 类型。此函数的结果与 Date 函数相同
Tomorrow	DateUtils	此函数可返回昨天的日期，其返回值为 TDateTime 类型
Yesterday	DateUtils	此函数可返回明天的日期，其返回值为 TDateTime 类型
CurrentYear	SysUtils	此函数可返回现在所属的年度，其返回值为 4 位的整数。例如:2001
HoursPerDay	SysUtils	此常数定义每天的小时数。 HoursPerDay = 24;
MinsPerDay	SysUtils	此常数定义每天的分钟数。 MinsPerDay = HoursPerDay * 60;
SecsPerDay	SysUtils	此常数定义每天的秒数。 SecsPerDay = MinsPerDay * 60;
MSecsPerDay	SysUtils	此常数定义每天的毫秒数。 MSecsPerDay = SecsPerDay * 1000;

以下将以范例的方式说明每一个函数的使用方法。

Now(返回当前的日期时间)

引用单元：SysUtils

函数声明：function Now: TDateTime;

范例 D-1

```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  MyDateTime := Now;
  Showmessage(DateTimeToStr(MyDateTime));
end;
```

- *Date(返回当前的日期)*

引用单元：SysUtils

函数声明：function Date: TDateTime;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  MyDateTime := Date;
  Showmessage(DateToStr(MyDateTime));
end;
```

- *Time(返回当前的时间)*

引用单元：SysUtils

函数声明：function Time: TDateTime;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  MyDateTime := Time;
  Showmessage(TimeToStr(MyDateTime));
end;
```

- *Today(返回今天的日期)*

引用单元：DateUtils

函数声明：function Today: TDateTime;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  //uses DateUtils
  MyDateTime := Today;
  Showmessage(DateToStr(MyDateTime)); //与 Date 函数相同的结果
end;
```

Tomorrow (返回明天的日期)

引用单元：DateUtils

函数声明：function Tomorrow: TDateTime;



D-5

```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  //uses DateUtils
  MyDateTime := Tomorrow;
  //MyDateTime := Now + 1; //两者相同
  Showmessage(DateToStr(MyDateTime)); //不包含时间部分
end;
```

Yesterday (返回昨天的日期)

引用单元：DateUtils

函数声明：function Yesterday: TDateTime;



D-6

```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  //uses DateUtils
  MyDateTime := Yesterday;
  //MyDateTime := Now - 1 ; //两者相同
  Showmessage(DateToStr(MyDateTime)); //不包含时间部分
end;
```

CurrentYear (返回现在所属的年度)

引用单元：SysUtils

函数声明：function CurrentYear: Word;



D-7

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ThisYear : Word;
begin
  ThisYear := CurrentYear;
```

```
  Showmessage(IntToStr(ThisYear)); //4位整数
end;
```

HoursPerDay、*MinsPerDay*、*SecsPerDay* 及 *MsecsPerDay* 等日期与时间常数。

引用单元：SysUtils

函数声明：function CurrentYear: Word;

范例 D-8

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Showmessage('每天的小时数=' + IntToStr(HoursPerDay)); //24
  Showmessage('每天的分钟数=' + IntToStr(MinsPerDay));  //1440
  Showmessage('每天的秒数=' + IntToStr(SecsPerDay));    //86400
  Showmessage('每天的毫秒数=' + IntToStr(MSecsPerDay));
    //86400000
end;
```

D.2 日期处理函数

本小节将为您介绍 Delphi 6 所提供的日期处理函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 D.2 所示。

表 D.2

函数名称	单元文件	所代表的意义
YearOf	DateUtils	此函数可获取 TDateTime 格式中的年度，其返回值为 Word 类型
YearsBetween	DateUtils	此函数可返回两个指定日期间的年份，一年以 365.25 天为计算单位，其返回值不包含小数部分
YearSpan	DateUtils	此函数可返回两个指定日期间的年份，一年以 365.25 天为计算单位，其返回值包含小数部分
StartOfAYear	DateUtils	此函数可返回特定年份的第一天
EndOfAYear	DateUtils	此函数可返回特定年份的最后一天
StartOfTheYear	DateUtils	此函数可返回特定日期的该年第一天
EndOfTheYear	DateUtils	此函数可返回特定日期的该年最后一天
IncYear	DateUtils	此函数可将指定的 TDateTime 变量加上指定的年度，其默认值为 1 年
MonthOf	DateUtils	此函数可获取 TDateTime 格式中的月份，其返回值为 Word 类型
MonthOfTheYear	DateUtils	此函数可获取 TDateTime 格式中的月份，其返回值为 Word 类型
MonthsBetween	DateUtils	此函数可返回两个指定日期间的月份数，一个月以 30.4375 天为计算单位，其返回值不包含小数部分
MonthSpan	DateUtils	此函数可返回两个指定日期间的月份数，一个月以 30.4375 天为计算单位，其返回值包含小数部分

续上表

函数名称	单元文件	所代表的意义
StartOfAMonth	DateUtils	此函数可返回特定年月的第一天
EndOfAMonth	DateUtils	此函数可返回特定年月的最后一天
StartOfTheMonth	DateUtils	此函数可返回指定日期的该年的第一天
EndOfTheMonth	DateUtils	此函数可返回指定日期的该年的最后一天
IncMonth	DateUtils	此函数可将指定的 TDateTime 变量加上指定的月份，其默认值为加上 1 个月
IncAMonth	DateUtils	此函数可将指定的年月日变量加上指定的月份，其默认值为加上 1 个月
DaysInAYear	DateUtils	此函数可返回指定年份的总天数
DaysInYear	DateUtils	此函数可返回指定 TDateTime 变量中该年份的总天数
DaysInAMonth	DateUtils	此函数可返回指定月份的总天数
DaysInMonth	DateUtils	此函数可返回指定 TDateTime 变量中该月份的总天数
DayOf	DateUtils	此函数可获取 TDateTime 格式中的日期，其返回值为 Word 类型
DaysBetween	DateUtils	此函数可返回两个指定日期间的天数，其返回值不包含小数部分
DaySpan	DateUtils	此函数可返回两个指定日期间的天数，其返回值包含小数部分
DayOfTheYear	DateUtils	此函数可返回指定 TDateTime 变量为该年的第几天。例如 2 月 1 日则返回 32
DayOfTheMonth	DateUtils	此函数可返回指定 TDateTime 变量为该月的第几天，其返回值介于 1 到 31
DayOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量为该周的第几天，其返回值介于 1 到 7。星期一为第一天
DayOfWeek	SysUtils	此函数可返回指定 TDateTime 变量为该周的第几天，其返回值介于 1 到 7。星期天为第一天
StartOfADay	DateUtils	此函数可返回指定日期一天的开始时间，其返回值为 TDateTime 类型。其时间默认为 12:00:000 AM
EndOfADay	DateUtils	此函数可返回指定日期一天的结束时间，其返回值为 TDateTime 类型。其时间默认为 11:59:999 PM
StartOfTheDay	DateUtils	此函数可返回指定 TDateTime 变量的一天开始时间，其返回值为 TDateTime 类型。其时间默认为 12:00:000 AM
EndOfTheDay	DateUtils	此函数可返回指定 TDateTime 变量的一天结束时间，其返回值为 TDateTime 类型。其时间默认为 11:59:999 PM
IncDay	DateUtils	此函数可为指定日期加上特定的天数，其返回值为 TDateTime 类型。其默认天数为 1 天
WeeksInAYear	DateUtils	此函数可返回指定年度的周数，其返回值不是 52 就是 53
WeeksInYear	DateUtils	此函数可返回指定 TDateTime 变量的周数，其返回值不是 52 就是 53

续上表

函数名称	单元文件	所代表的意义
WeekOf	DateUtils	此函数可返回指定日期为该年的第几周，其返回值为 1~53
WeekOfTheYear	DateUtils	此函数可返回指定日期为该年的第几周，其返回值为 1~53
WeekOfTheMonth	DateUtils	此函数可返回指定日期为该月的第几周，其返回值为 1~6
WeeksBetween	DateUtils	此函数可返回两个指定日期间的周数，其返回值不包含小数部分
WeekSpan	DateUtils	此函数可返回两个指定日期间的周数，其返回值包含小数部分
StartOfAWeek	DateUtils	此函数可返回指定日期一周的开始时间，其返回值为 TDateTime 类型。其时间默认为 12:00:000 AM
EndOfAWeek	DateUtils	此函数可返回指定日期一周的结束时间，其返回值为 TDateTime 类型其时间默认为 11:59:999 PM
StartOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量的一周开始时间，其返回值为 TDateTime 类型。其时间默认为 12:00:000 AM
EndOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量的一周结束时间，其返回值为 TDateTime 类型。其时间默认为 11:59:999 PM
IncWeek	DateUtils	此函数可将指定日期加上指定周数，其返回值为 TDateTime 类型

以下将以范例的方式说明每一个函数的使用方法。

YearOf(返回指定日期的年度)

引用单元：DateUtils

函数声明：function YearOf(const AValue: TDateTime): Word;

范例 D-9

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //三者皆相同
  Showmessage('年度=' + IntToStr(YearOf(Now)));
  Showmessage('年度=' + IntToStr(YearOf(Date)));
  Showmessage('年度=' + IntToStr(YearOf(Today)));
end;
```

YearsBetween(返回两个指定日期间的年份)

引用单元：DateUtils

函数声明：function YearsBetween(const ANow, AThen: TDateTime): Integer;

范例 D-10

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  //不包含小数，一年以 365.25 天为计算单位
  Showmessage('几年=' + IntToStr(YearsBetween(Now,Now+560))); //1
end;
```

- *YearSpan(返回两个指定日期间的年份)*

引用单元：DateUtils

函数声明：function YearSpan(const ANow, AThen: TDateTime): Double;

例 D-11

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //包含小数，一年以 365.25 天为计算单位
  Showmessage('几年=' + FloatToStr(YearSpan (Now,Now+560)));
        //1.53...
end;
```

- *StartOfAYear(返回特定年份的第一天)*

引用单元：DateUtils

函数声明：function StartOfAYear(const AYear): TDateTime;

例 D-12

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Showmessage(DateTimeToStr(StartOfAYear(2001)));
  //2001/1/1 早上 12:00:00
end;
```

- *EndOfAYear(返回特定年份的最后一天)*

引用单元：DateUtils

函数声明：function EndOfAYear(const AYear): TDateTime;

例 D-13

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Showmessage(DateTimeToStr(EndOfAYear(2001)));
  //2001/12/31 下午 11:59:59
end;
```

- *StartOfTheYear*(返回指定日期的该年的第一天)

引用单元：DateUtils
函数声明：function StartOfTheYear(const AValue: TDateTime): TDateTime;

范例 D-14

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Showmessage(DateTimeToStr(StartOfTheYear(Now)));
  //指定的年度/1/1 早上 12:00:00
end;
```

- *EndOfTheYear*(返回指定日期的该年的最后一天)

引用单元：DateUtils
函数声明：function EndOfTheYear(const AValue: TDateTime): TDateTime;

范例 D-15

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Showmessage(DateTimeToStr(EndOfTheYear(Now)));
  //指定年度/12/31下午 11:59:59
end;
```

- *IncYear*(将指定的 *TDateTime* 变量加上指定的年)

引用单元：DateUtils
函数声明：function IncYear(const AValue: TDateTime; const ANumberOfYears: Integer = 1): TDateTime;

范例 D-16

```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
begin
  myDateTime := IncYear(Now,3); //往后加3年
  Showmessage(DateTimeToStr(myDateTime));
end;
```

- *MonthOf*(获取 *TDateTime* 格式中的月份)

引用单元：DateUtils
函数声明：function MonthOf(const AValue: TDateTime): Word;

- *MonthOfTheYear(获取 TDateTime 格式中的月份)*

引用单元：DateUtils
函数声明：function MonthOfTheYear(const AValue: TDateTime): Word;



D-17

```
procedure TForm1.Button1Click(Sender: TObject);
var
  myDateTime : TDateTime;
  m : Integer;
begin
  myDateTime := Now;
  //两个函数获取相同的结果
  m  := MonthOf(myDateTime);
  Showmessage(IntToStr(m));
  m  := MonthOfTheYear(myDateTime);
  Showmessage(IntToStr(m));
end;
```

- *MonthsBetween(返回两个指定日期间的月份数)*

引用单元：DateUtils
函数声明：function MonthsBetween(const ANow, AThen: TDateTime): Integer;

- *MonthSpan(返回两个指定日期间的月份数)*

引用单元：DateUtils
函数声明：function MonthSpan(const ANow, AThen: TDateTime): Double;



D-18

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  f : Double;
begin
  i  := MonthsBetween(Now,Now+89); //差89天
  Showmessage(IntToStr(i)); //2
  f  := MonthSpan(Now,Now+89); //差89天
  Showmessage(FloatToStr(f)); //2.924....
end;
```

- *StartOfAMonth*(返回特定年月的第一天)

引用单元：DateUtils
函数声明：function StartOfAMonth(const AYear, AMonth: Word): TDateTime;

- *EndOfAMonth*(返回特定年月的最后一天)

引用单元：DateUtils
函数声明：function EndOfAMonth(const AYear, AMonth: Word): TDateTime;

- *StartOfTheMonth*(返回特定年月的第一天)

引用单元：DateUtils
函数声明：function StartOfTheMonth(const AValue: TDateTime): TDateTime;

- *EndOfTheMonth*(返回特定年月的最后一天)

引用单元：DateUtils
函数声明：function EndOfTheMonth(const AValue: TDateTime): TDateTime;



D-19

```
var
  myDateTime : TDateTime;
  Year, Month : Integer;
begin
  Year := YearOf(Now);
  Month := MonthOf(Now);
  //找出当前月份的第一天
  myDateTime := StartOfAMonth(Year,Month);
  Showmessage(DateTimeToStr(myDateTime));
  //找出当前月份的最后一天
  myDateTime := EndOfAMonth(Year,Month);
  Showmessage(DateTimeToStr(myDateTime));
  //找出当前月份的第一天
  myDateTime := StartOfTheMonth(Now);
  Showmessage(DateTimeToStr(myDateTime));
  //找出当前月份的最后一天
  myDateTime := EndOfTheMonth(Now);
  Showmessage(DateTimeToStr(myDateTime));
end;
```

- *IncMonth*(指定的 *TDateTime* 变量加上指定的月份)

引用单元：DateUtils
函数声明：function IncMonth(const Date: TDateTime; NumberOfMonths: Integer = 1): TDateTime;

- *IncAMonth(指定的年月日变量加上指定的月份)*

引用单元：DateUtils
函数声明：procedure IncAMonth(var Year, Month, Day: Word; NumberOfMonths: Integer = 1);

范例 D-20

```
var
  myDateTime : TDateTime;
  Year, Month, Day : Word;
begin
  //加上指定的月份
  myDateTime := IncMonth(Now,3); //加上三个月
  Showmessage(DateTimeToStr(myDateTime));

  Year := YearOf(Now);
  Month := MonthOf(Now);
  Day := DayOf(Now);
  //加上指定的月份
  IncAMonth(Year, Month, Day, 3);
  Showmessage('Month=' + IntToStr(Month));
end;
```

- *DaysInAYear(返回指定年份的总天数)*

引用单元：DateUtils
函数声明：function DaysInAYear(const AYear: Word): Word;

- *DaysInYear(返回指定 TDateTime 变量中该年份的总天数)*

引用单元：DateUtils
函数声明：function DaysInYear(const AValue: TDateTime): Word;

- *DaysInAMonth(返回指定月份的总天数)*

引用单元：DateUtils
函数声明：function DaysInAMonth(const AYear, AMonth: Word): Word;

- *DaysInMonth(返回指定 TDateTime 变量中该月份的总天数)*

引用单元：DateUtils
函数声明：function DaysInMonth(const AValue: TDateTime): Word;

范例 D-21

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  myDateTime : TDateTime;
  Year, Month : Word;
  i : Integer;
begin
  Year := YearOf(Now);
  //获取指定年份的总天数
  i := DaysInAYear(Year);
  Showmessage('DaysInAYear=' + IntToStr(i) );
  i := DaysInYear(now);
  Showmessage('DaysInYear=' + IntToStr(i) );
  //获取指定月份的总天数
  Month := MonthOf(Now);
  i := DaysInAMonth(Year,Month);
  Showmessage('DaysInAMonth=' + IntToStr(i) );
  i := DaysInMonth(Now);
  Showmessage('DaysInMonth=' + IntToStr(i) );
end;
```

- *DayOf(获取 TDateTime 格式中的日期)*

引用单元：DateUtils
函数声明：function DayOf(const AValue: TDateTime): Word;

范例 D-22

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Day : Word;
begin
  //获取指定 TDateTime 的日期
  Day := DayOf(Now);
  Showmessage(IntToStr(Day));
end;
```

- *DaysBetween(返回两个指定日期间的天数)*

引用单元：DateUtils
函数声明：function DaysBetween(const ANow, AThen: TDateTime): Integer;

- *DaySpan(返回两个指定日期间的天数)*

引用单元：DateUtils
函数声明：function DaySpan(const ANow, AThen: TDateTime): Double;

范例 D-23

```
var
```

```
  i : Integer;
  f : Double;
begin
  //获取两个日期的天数
  i := DaysBetween(Now-500,Now);
  Showmessage(IntToStr(i)); //500
  //获取两个日期的天数
  f := DaySpan(Now-500,Now);
  Showmessage(FloatToStr(f)); //500
end;
```

- *DayOfTheYear(返回指定 TDateTime 变量为该年的第几天)*

引用单元：DateUtils
函数声明：function DayOfTheYear(const AValue: TDateTime): Word;

- *DayOfTheMonth(返回指定 TDateTime 变量为该月的第几天)*

引用单元：DateUtils
函数声明：function DayOfTheMonth(const AValue: TDateTime): Word;

- *DayOfTheWeek(返回指定 TDateTime 变量为该周的第几天)*

引用单元：DateUtils
函数声明：function DayOfTheWeek(const AValue: TDateTime): Word;

- *DayOfWeek(返回指定 TDateTime 变量为该周的第几天)*

引用单元：SysUtils
函数声明：function DayOfWeek(Date: TDateTime): Integer;

范例 D-24

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
begin
  //返回指定 TDateTime 为该年的第几天
  i := DayOfTheYear(Now);
  Showmessage(IntToStr(j));
  //返回指定 TDateTime 为该月的第几天
  i := DayOfTheMonth (Now);
  Showmessage(IntToStr(i));
  //返回指定 TDateTime 为该周的第几天(星期一为第一天)
  i := DayOfTheWeek(Now);
```

```
  Showmessage(IntToStr(i));
  //返回指定 TDateTime 为该周的第几天(星期日为第一天)
  i := DayOfWeek(Now);
  Showmessage(IntToStr(i));
end;
```

StartOfADay(返回指定日期一天的开始时间)

引用单元：DateUtils
函数声明：function StartOfADay(const AYear, AMonth, ADay: Word): TDateTime; function StartOfADay(const AYear, ADayOfYear: Word): TDateTime;

范例 D-25

```
procedure TForm1.Button1Click(Sender: TObject);
var
  tmpDateTime : TDateTime;
  AYear, ADayOfYear, AMonth, ADay: Word;
begin
  AYear := 2001;    //指定年度
  ADayOfYear := 32; //第 32 天
  tmpDateTime := StartOfADay(AYear,ADayOfYear);
  Showmessage(DateTimeToStr(tmpDateTime)); //2001/02/01
  AMonth := 1;
  ADay := 32; //从 AMonth 第一天起算 32 天
  tmpDateTime := StartOfADay(AYear,AMonth,ADay);
  Showmessage(DateTimeToStr(tmpDateTime)); //2001/02/01
end;
```

EndOfADay(返回指定日期一天的结束时间)

引用单元：DateUtils
函数声明：function EndOfADay(const AYear, AMonth, ADay: Word): TDateTime; function EndOfADay(const AYear, ADayOfYear: Word): TDateTime;

范例 D-26

```
procedure TForm1.Button1Click(Sender: TObject);
var
  tmpDateTime : TDateTime;
  AYear, ADayOfYear, AMonth, ADay: Word;
begin
  AYear := 2001;    //指定年度
  ADayOfYear := 32; //第 32 天
  tmpDateTime := EndOfADay(AYear,ADayOfYear);
```

```
  Showmessage(DateTimeToStr(tmpDateTime));
  //2001/07/01 PM 11:59:59
  AMonth := 1;
  ADay := 32; //从AMonth第一天起算32天
  tmpDateTime := EndOfADay(AYear,AMonth,ADay);
  Showmessage(DateTimeToStr(tmpDateTime));
  //2001/02/01 PM 11:59:59
end;
```

- *StartOfTheDay(返回指定 TDateTime 变量的一天开始时间)*

引用单元：DateUtils
函数声明：function StartOfTheDay(const AValue: TDateTime): TDateTime;

- *EndOfTheDay(返回指定 TDateTime 变量的一天结束时间)*

引用单元：DateUtils
函数声明：function EndOfTheDay(const AValue: TDateTime): TDateTime;

范例 D-27

```
procedure TForm1.Button1Click(Sender: TObject);
var
  tmpDateTime : TDateTime;
begin
  tmpDateTime := StartOfTheDay(Now);
  Showmessage(DateTimeToStr(tmpDateTime));
  //YYYY/MM/DD AM 12:00:00
  tmpDateTime := EndOfTheDay(Now);
  Showmessage(DateTimeToStr(tmpDateTime));
  //YYYY/MM/DD PM 11:59:59
end;
```

- *IncDay(可为指定日期加上特定的天数)*

引用单元：DateUtils
函数声明：function IncDay(const AValue: TDateTime; const ANumberOfDays: Integer = 1): TDateTime;

范例 D-28

```
procedure TForm1.Button1Click(Sender: TObject);
var
  tmpDateTime : TDateTime;
begin
```

```
  tmpDateTime := IncDay(Now,5);
  Showmessage(DateTimeToStr(tmpDateTime));
  //时间部分并不会改变
end;
```

WeeksInAYear(返回指定年度的周数)

引用单元：DateUtils
函数声明：function WeeksInAYear(const AYear: Word): Word;

WeeksInYear(返回指定 TDateTime 变量的周数)

引用单元：DateUtils
函数声明：function WeeksInYear(const AValue: TDateTime): Word;

范例 D-29

```
procedure TForm1.Button1Click(Sender: TObject);
var
  weeks : word;
begin
  weeks := WeeksInAYear(2001);
  Showmessage(IntToStr(weeks)); //52
  weeks := WeeksInYear(Now);
  Showmessage(IntToStr(weeks)); //52
end;
```

WeekOf(返回指定日期为该年的第几周)

引用单元：DateUtils
函数声明：function WeekOf(const AValue: TDateTime): Word;

WeekOfTheYear(返回指定日期为该年的第几周)

引用单元：DateUtils
函数声明：function WeekOfTheYear(const AValue: TDateTime): Word;

WeekOfTheMonth(返回指定日期为该月的第几周)

引用单元：DateUtils
函数声明：function WeekOfTheMonth(const AValue: TDateTime): Word;

范例 D-30

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  i : word;
begin
  i := WeekOf(Now);
  Showmessage('WeekOf=' + IntToStr(i));
  i := WeekOfTheYear(Now);
  Showmessage('WeekOfTheYear=' + IntToStr(i));
  i := WeekOfTheMonth(Now);
  Showmessage('WeekOfTheMonth=' + IntToStr(i));
end;
```

- *WeeksBetween*(返回两个指定日期间的周数)

引用单元：DateUtils
函数声明：function WeeksBetween(const ANow, AThen: TDateTime): Integer;

- *WeekSpan*(返回两个指定日期间的周数)

引用单元：DateUtils
函数声明：function WeekSpan(const ANow, AThen: TDateTime): Double;

范例 D-31

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  f : Double;
begin
  i := WeeksBetween(Now,Now+29);
  Showmessage(IntToStr(i)); //4
  f := WeekSpan(Now,Now+29);
  Showmessage(FloatToStr(f)); //4.14...
end;
```

- *StartOfAWeek*(返回指定日期一周的开始时间)

引用单元：DateUtils
函数声明：function StartOfAWeek(const AYear, AWeekOfYear: Word; const ADayOfWeek: Word = 1): TDateTime;

- *EndOfAWeek*(返回指定日期一周的结束时间)

引用单元：DateUtils
函数声明：function EndOfAWeek(const AYear, AWeekOfYear: Word; const ADayOfWeek: Word = 7): TDateTime;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  tempDateTime : TDateTime;
  AYear, AWeekOfYear, ADayOfWeek : word;
begin
  AYear := 2001;
  AWeekOfYear := 2; //第二周
  ADayOfWeek := 1;  //第一天
tempDateTime:=StartOfAWeek(AYear,AWeekOfYear,ADayOfWeek);
  //2001/01/08 AM 12:00:00:00
  Showmessage(DateTimeToStr(tempDateTime));
  //AYear := 2001;
  //AWeekOfYear := 2; //第二周
  ADayOfWeek := 7;  //第七天
tempDateTime := EndOfAWeek(AYear,AWeekOfYear,ADayOfWeek);
  //2001/01/14 PM 11:59:59
  Showmessage(DateTimeToStr(tempDateTime));
end;
```

- *StartOfTheWeek(返回指定日期一周的开始时间)*

引用单元：DateUtils
函数声明：function StartOfTheWeek(const AValue: TDateTime): TDateTime;

- *EndOfTheWeek(返回指定日期一周的结束时间)*

引用单元：DateUtils
函数声明：function EndOfTheWeek(const AValue: TDateTime): TDateTime;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  tempDateTime : TDateTime;
begin
  tempDateTime := StartOfTheWeek(Now);
  //YYYY/MM/DD AM 12:00:00:00
  Showmessage(DateTimeToStr(tempDateTime));
  tempDateTime := EndOfTheWeek(Now);
  //YYYY/MM/DD+6 PM 11:59:59
  Showmessage(DateTimeToStr(tempDateTime));
end;
```

IncWeek(将指定日期加上指定周数)

引用单元：DateUtils
函数声明：function IncWeek(const AValue: TDateTime; const ANumberOfWeeks: Integer = 1): TDateTime;

范例 D-34

```
procedure TForm1.Button1Click(Sender: TObject);
var
  tempDateTime : TDateTime;
begin
  tempDateTime := IncWeek(Now,5);//加上 5 周
  Showmessage(DateTimeToStr(tempDateTime));
end;
```

D.3 时间处理函数

本小节将为您介绍 Delphi 6 所提供的时间处理函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 D.3 所示。

表 D.3

函数名称	单元文件	所代表的意义
HoursBetween	DateUtils	此函数可返回两个指定 TDateTime 变量间的小时数，返回值不包含小数部分
HourSpan	DateUtils	此函数可返回两个指定 TDateTime 变量间的小时数，返回值包含小数部分
HourOf	DateUtils	此函数可获取 TDateTime 格式中的小时数，其返回值为 Word 类型，介于 0～23 之间
HourOfTheYear	DateUtils	此函数可返回指定 TDateTime 变量为该年的第几小时
HourOfTheMonth	DateUtils	此函数可返回指定 TDateTime 变量为该月的第几小时
HourOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量为该周的第几小时
HourOfTheDay	DateUtils	此函数可返回指定 TDateTime 变量为该日的第几小时，返回值介于 0～23 之间
IncHour	DateUtils	此函数可将指定日期加上指定小时数，其返回值为 TDateTime 类型
MinutesBetween	DateUtils	此函数可返回两个指定 TDateTime 变量间的分钟数，返回值不包含小数部分
MinuteSpan	DateUtils	此函数可返回两个指定 TDateTime 变量间的分钟数，返回值包含小数部分

续上表

函数名称	单元文件	所代表的意义
MinuteOf	DateUtils	此函数可获取 TDateTime 格式中的分钟数，其返回值为 Word 类型，介于 0～59 之间
MinuteOfTheYear	DateUtils	此函数可返回指定 TDateTime 变量为该年的第几分钟
MinuteOfTheMonth	DateUtils	此函数可返回指定 TDateTime 变量为该月的第几分钟
MinuteOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量为该周的第几分钟
MinuteOfTheDay	DateUtils	此函数可返回指定 TDateTime 变量为该日的第几分钟
MinuteOfTheHour	DateUtils	此函数可返回指定 TDateTime 变量为该小时的第几分钟
IncMinute	DateUtils	此函数可将指定日期加上指定分钟数，其返回值为 TDateTime 类型
SecondsBetween	DateUtils	此函数可返回两个指定 TDateTime 变量间的秒数，返回值不包含小数部分
SecondSpan	DateUtils	此函数可返回两个指定 TDateTime 变量间的秒数，返回值包含小数部分
SecondOf	DateUtils	此函数可获取 TDateTime 格式中的秒数，其返回值为 Word 类型，介于 0～59 之间
SecondOfTheYear	DateUtils	此函数可返回指定 TDateTime 变量为该年的第几秒
SecondOfTheMonth	DateUtils	此函数可返回指定 TDateTime 变量为该月的第几秒
SecondOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量为该周的第几秒
SecondOfTheDay	DateUtils	此函数可返回指定 TDateTime 变量为该日的第几秒
SecondOfTheHour	DateUtils	此函数可返回指定 TDateTime 变量为该小时的第几秒
SecondOfTheMinute	DateUtils	此函数可返回指定 TDateTime 变量为该分钟的第几秒
IncSecond	DateUtils	此函数可将指定日期加上指定秒数，其返回值为 TDateTime 类型
MilliSecondsBetween	DateUtils	此函数可返回两个指定 TDateTime 变量间的毫秒数，返回值不包含小数部分
MilliSecondSpan	DateUtils	此函数可返回两个指定 TDateTime 变量间的毫秒数，返回值包含小数部分
MilliSecondOf	DateUtils	此函数可获取 TDateTime 格式中的毫秒数，其返回值为 Word 类型，介于 0～999 之间
MilliSecondOfTheYear	DateUtils	此函数可返回指定 TDateTime 变量为该年的第几毫秒
MilliSecondOfTheMonth	DateUtils	此函数可返回指定 TDateTime 变量为该月的第几毫秒
MilliSecondOfTheWeek	DateUtils	此函数可返回指定 TDateTime 变量为该周的第几毫秒
MilliSecondOfTheDay	DateUtils	此函数可返回指定 TDateTime 变量为该日的第几毫秒
MilliSecondOfTheHour	DateUtils	此函数可返回指定 TDateTime 变量为该小时的第几毫秒
MilliSecondOfTheMinute	DateUtils	此函数可返回指定 TDateTime 变量为该分钟的第几毫秒

续上表

函数名称	单元文件	所代表的意义
MilliSecondOfTheSecond	DateUtils	此函数可返回指定 TDateTime 变量为该秒的第几毫秒
IncMilliSecond	DateUtils	此函数可将指定日期加上指定毫秒数，其返回值为 TDateTime 类型

以下将以范例的方式说明每一个函数的使用方法。

- *HoursBetween(返回两个指定日期间的小时数)*

引用单元：DateUtils
函数声明：function HoursBetween(const ANow, AThen: TDateTime): Int64;

- *HourSpan(返回两个指定日期间的小时数)*

引用单元：DateUtils
函数声明：function HourSpan(const ANow, AThen: TDateTime): Double;

范例 D-35

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Hours : Integer;
  f : Double;
begin
  Hours := HoursBetween(Now,Now+0.9);//加上 0.9 天
  Showmessage(IntToStr(Hours)); //21

  f := HourSpan(Now,Now+0.9);//加上 0.9 天
  Showmessage(FloatToStr(f)); //21.6....
end;
```

- *HourOf(获取 TDateTime 格式中的时数)*

引用单元：DateUtils
函数声明：function HourOf(const AValue: TDateTime): Word;

- *HourOfTheYear(返回指定 TDateTime 变量为该年的第几小时)*

引用单元：DateUtils
函数声明：function HourOfTheYear(const AValue: TDateTime): Word;

- *HourOfTheMonth(返回指定 TDateTime 变量为该月的第几小时)*

引用单元：DateUtils
函数声明：function HourOfTheMonth(const AValue: TDateTime): Word;

- *HourOfTheWeek(返回指定 TDateTime 变量为该周的第几小时)*

引用单元：DateUtils
函数声明：function HourOfTheWeek(const AValue: TDateTime): Word;

- *HourOfTheDay(返回指定 TDateTime 变量为该日的第几小时)*

引用单元：DateUtils
函数声明：function HourOfTheDay(const AValue: TDateTime): Word;



D-36

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Hours : Integer;
begin
  Hours := HourOf(Now);
  Showmessage('HourOf=' + IntToStr(Hours));
  Hours := HourOfTheYear(Now);
  Showmessage('HourOfTheYear=' + IntToStr(Hours));
  Hours := HourOfTheMonth(Now);
  Showmessage('HourOfTheMonth=' + IntToStr(Hours));
  Hours := HourOfTheWeek(Now);
  Showmessage('HourOfTheWeek=' + IntToStr(Hours));
  Hours := HourOfTheDay(Now); //与 HourOf 函数的返回值相同
  Showmessage('HourOfTheDay=' + IntToStr(Hours));
end;
```

- *IncHour(将指定日期加上指定小时数)*

引用单元：DateUtils
函数声明：function IncHour(const AValue: TDateTime; const ANumberOfHours: Int64 = 1): TDateTime;



D-37

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TempDateTime : TDateTime;
begin
  TempDateTime := IncHour(Now,23); //增加 23 小时
  Showmessage(DateTimeToStr(TempDateTime));
end;
```

- *MinutesBetween*(返回两个指定 *TDateTime* 变量间的分钟数)

引用单元：DateUtils
函数声明：function MinutesBetween(const ANow, AThen: TDateTime): Int64;

- *MinuteSpan*(返回两个指定 *TDateTime* 变量间的分钟数)

引用单元：DateUtils
函数声明：function MinuteSpan(const ANow, AThen: TDateTime): Double;

- *MinuteOf*(获取 *TDateTime* 格式中的分钟数)

引用单元：DateUtils
函数声明：function MinuteOf(const AValue: TDateTime): Word;

例 D-38

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  f : Double;
  w : word;
begin
  i := MinutesBetween(Now,Now + 1); //增加 24 小时
  Showmessage(IntToStr(i)); //1440

  f := MinuteSpan(Now, Now- 1.1); //增加 24*1.1 小时
  Showmessage(FloatToStr(f)); //1583.99.....

  w := MinuteOf(Now);       //显示当前的分钟数
  Showmessage(IntToStr(w)); //介于 0-59 之间
end;
```

- *MinuteOfTheYear*(返回指定 *TDateTime* 变量为该年的第几分钟)

引用单元：DateUtils
函数声明：function MinuteOfTheYear(const AValue: TDateTime): LongWord;

- *MinuteOfTheMonth*(返回指定 *TDateTime* 变量为该月的第几分钟)

引用单元：DateUtils
函数声明：function MinuteOfTheMonth(const AValue: TDateTime): Word;

- *MinuteOfTheWeek*(返回指定 *TDateTime* 变量为该周的第几分钟)

引用单元：DateUtils
函数声明：function MinuteOfTheWeek(const AValue: TDateTime): Word;

MinuteOfTheDay(返回指定 *TDateTime* 变量为该日的第几分钟)

引用单元：DateUtils
函数声明：function MinuteOfTheDay(const AValue: TDateTime): Word;

MinuteOfTheHour(返回指定 *TDateTime* 变量为该小时的第几分钟)

引用单元：DateUtils
函数声明：function MinuteOfTheHour(const AValue: TDateTime): Word;

范例 D-39

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Minutes : Integer;
begin
  Minutes := MinuteOfTheYear(Now);
  ShowMessage('MinuteOfTheYear=' + IntToStr(Minutes));
  Minutes := MinuteOfTheMonth(Now);
  ShowMessage('MinuteOfTheMonth=' + IntToStr(Minutes));
  Minutes := MinuteOfTheWeek(Now);
  ShowMessage('MinuteOfTheWeek=' + IntToStr(Minutes));
  Minutes := MinuteOfTheDay(Now); //与 MinutesOf 函数返回值相同
  ShowMessage('MinuteOfTheDay=' + IntToStr(Minutes));
  Minutes := MinuteOfTheHour(Now);
  ShowMessage('MinuteOfTheHour=' + IntToStr(Minutes));
end;
```

IncMinute(将指定日期加上指定分钟数)

引用单元：DateUtils
函数声明：function IncMinute(const AValue: TDateTime; const ANumberOfMinutes: Int64 = 1): TDateTime;

范例 D-40

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TempDateTime : TDateTime;
begin
  TempDateTime := IncMinute(Now,40); //增加 40 分钟
  ShowMessage(DateTimeToStr(TempDateTime));
end;
```

- *SecondsBetween*(返回两个指定 *TDateTime* 变量间的秒数)

引用单元：DateUtils
函数声明：function SecondsBetween(const ANow, AThen: TDateTime): Int64;

- *SecondSpan*(返回两个指定 *TDateTime* 变量间的秒数)

引用单元：DateUtils
函数声明：function SecondSpan(const ANow, AThen: TDateTime): Double;

- *SecondOf*(获取 *TDateTime* 格式中的秒数)

引用单元：DateUtils
函数声明：function SecondOf(const AValue: TDateTime): Word;

D-41

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  f : Double;
  w : word;
begin
  i := SecondsBetween(Now,Now + 1); //增加 24 小时
  Showmessage(IntToStr(i)); //86400

  f := SecondSpan(Now, Now- 1.1); //增加 24*1.1 小时
  Showmessage(FloatToStr(f)); //95039.99.....

  w := SecondOf(Now);       //显示当前的秒数
  Showmessage(IntToStr(w)); //介于 0~59 之间
end;
```

- *SecondOfTheYear*(返回指定 *TDateTime* 变量为该年的第几秒)

引用单元：DateUtils
函数声明：function SecondOfTheYear(const AValue: TDateTime): LongWord;

- *SecondOfTheMonth*(返回指定 *TDateTime* 变量为该月的第几秒)

引用单元：DateUtils
函数声明：function SecondOfTheMonth(const AValue: TDateTime): LongWord;

- *SecondOfTheWeek*(返回指定 *TDateTime* 变量为该周的第几秒)

引用单元：DateUtils
函数声明：function SecondOfTheWeek(const AValue: TDateTime): LongWord;

SecondOfTheDay(返回指定 TDateTime 变量为该日的第几秒)

引用单元：DateUtils
函数声明：function SecondOfTheDay(const AValue: TDateTime): LongWord;

SecondOfTheHour(返回指定 TDateTime 变量为该小时的第几秒)

引用单元：DateUtils
函数声明：function SecondOfTheHour(const AValue: TDateTime): Word;

SecondOfTheMinute(返回指定 TDateTime 变量为该分钟的第几秒)

引用单元：DateUtils
函数声明：function SecondOfTheMinute(const AValue: TDateTime): Word;

例 D-42

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Seconds : Integer;
begin
  Seconds := SecondOfTheYear(Now);
  Showmessage('SecondOfTheYear=' + IntToStr(Seconds));
  Seconds := SecondOfTheMonth(Now);
  Showmessage('SecondOfTheMonth=' + IntToStr(Seconds));
  Seconds := SecondOfTheWeek(Now);
  Showmessage('SecondOfTheWeek=' + IntToStr(Seconds));
  Seconds := SecondOfTheDay(Now); //与 SecondsOf 函数返回值相同
  Showmessage('SecondOfTheDay=' + IntToStr(Seconds));
  Seconds := SecondOfTheHour(Now);
  Showmessage('SecondOfTheHour=' + IntToStr(Seconds));
  Seconds := SecondOfTheMinute(Now);
  Showmessage('SecondOfTheMinute=' + IntToStr(Seconds));
end;
```

IncSecond(将指定日期加上指定秒数)

引用单元：DateUtils
函数声明：function IncSecond(const AValue: TDateTime; const ANumberOfSeconds: Int64 = 1): TDateTime;

例 D-43

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  TempDateTime : TDateTime;
begin
  TempDateTime := IncSecond(Now,240); //增加240秒
  Showmessage(DateTimeToStr(TempDateTime));
end;
```

- *MilliSecondsBetween(返回两个指定 TDateTime 变量间的毫秒数)*

引用单元：DateUtils
函数声明：function MilliSecondsBetween(const ANow, AThen: TDateTime): Int64;

- *MilliSecondSpan(返回两个指定 TDateTime 变量间的毫秒数)*

引用单元：DateUtils
函数声明：function MilliSecondSpan(const ANow, AThen: TDateTime): Double;

- *MilliSecondOf(获取 TDateTime 格式中的毫秒数)*

引用单元：DateUtils
函数声明：function MilliSecondOf(const AValue: TDateTime): Word;



D-44

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  f : Double;
  w : word;
begin
  i := MilliSecondsBetween(Now,Now + 0.1); //增加0.1小时
  Showmessage(IntToStr(i)); //863999

  f := MilliSecondSpan(Now, Now+ 0.111); //增加24*0.111小时
  Showmessage(FloatToStr(f)); //9590399.999....

  w := MilliSecondOf(Now);       //显示当前的毫秒数
  Showmessage(IntToStr(w)); //介于0~999之间
end;
```

- *MilliSecondOfTheYear(返回指定 TDateTime 变量为该年的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheYear(const AValue: TDateTime): Int64;

- *MilliSecondOfTheMonth(返回指定 TDateTime 变量为该月的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheMonth(const AValue: TDateTime): LongWord;

- *MilliSecondOfTheWeek(返回指定 TDateTime 变量为该周的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheWeek(const AValue: TDateTime): LongWord;

- *MilliSecondOfTheDay(返回指定 TDateTime 变量为该日的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheDay(const AValue: TDateTime): LongWord;

- *MilliSecondOfTheHour(返回指定 TDateTime 变量为该小时的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheHour(const AValue: TDateTime): LongWord;

- *MilliSecondOfTheMinute(返回指定 TDateTime 变量为该分钟的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheMinute(const AValue: TDateTime): LongWord;

- *MilliSecondOfTheSecond(返回指定 TDateTime 变量为该秒的第几毫秒)*

引用单元：DateUtils
函数声明：function MilliSecondOfTheSecond(const AValue: TDateTime): Word;

例 D-45

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MilliSeconds : Int64;
begin
  MilliSeconds := MilliSecondOfTheYear(Now);
  Showmessage('MilliSecondOfTheYear=' + IntToStr(MilliSeconds));
  MilliSeconds := MilliSecondOfTheMonth(Now);
  Showmessage('MilliSecondOfTheMonth=' + IntToStr(MilliSeconds));
  MilliSeconds := MilliSecondOfTheWeek(Now);
  Showmessage('MilliSecondOfTheWeek=' + IntToStr(MilliSeconds));
  MilliSeconds := MilliSecondOfTheDay(Now);
  //与MilliSecondsOf函数的返回值相同
  Showmessage('MilliSecondOfTheDay=' + IntToStr(MilliSeconds));
  MilliSeconds := MilliSecondOfTheHour(Now);
```

```
  Showmessage('MilliSecondOfTheHour=' + IntToStr(MilliSeconds));
  MilliSeconds := MilliSecondOfTheMinute(Now);
  Showmessage('MilliSecondOfTheMinute=' + IntToStr(MilliSeconds));
  MilliSeconds := MilliSecondOfTheSecond(Now);
  Showmessage('MilliSecondOfTheSecond=' + IntToStr(MilliSeconds));
end;
```

IncMilliSecond(将指定日期加上指定毫秒数)

引用单元：DateUtils

函数声明：function IncMilliSecond(const AValue: TDateTime; const AnumberOfMilliSeconds: Int64 = 1): TDateTime;

范例 D-46

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TempDateTime : TDateTime;
begin
  TempDateTime := IncMilliSecond(Now,24000);  //增加24000毫秒
  Showmessage(DateTimeToStr(TempDateTime));
end;
```

D.4 比较与判断函数

本小节将为您介绍 Delphi 6 所提供日期时间的比较与判断函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 D.4 所示。

表 D.4

函数名称	单元文件	所代表的意义
CompareDate	DateUtils	此函数可返回两个 TDateTime 变量的比较结果，此函数的比较对象为日期部分。返回值为－1、0 与 1，其代表的常数分别为 LessThanValue、EqualsValue 与 GreaterThanValue，当第一个变量值小于第二个变量值则返回－1；当两个变量值相等时则返回 0；当第一个变量值大于第二个变量值则返回 1
CompareTime	DateUtils	此函数可返回两个 TDateTime 变量的比较结果，此函数的比较对象为时间部分。返回值为－1、0 与 1，其代表的常数分别为 LessThanValue、EqualsValue 与 GreaterThanValue。当第一个变量值小于第二个变量值则返回－1；当两个变量值相等时则返回 0；当第一个变量值大于第二个变量值则返回 1

续上表

函数名称	单元文件	所代表的意义
CompareDateTime	DateUtils	此函数可返回两个 TDateTime 变量的比较结果，此函数的比较对象为完整的 TDateTime 变量。返回值为－1、0 与 1，其代表的常数分别为 LessThanValue、EqualsValue 与 GreaterThanValue。当第一个变量值小于第二个变量值则返回－1；当两个变量值相等时则返回 0；当第一个变量值大于第二个变量值则返回 1
SameDate	DateUtils	此函数可返回两个 TDateTime 变量的比较结果，此函数的比较对象为日期部分。返回值为布尔值，相等返回 True，否则，返回 False
SameTime	DateUtils	此函数可返回两个 TDateTime 变量的比较结果，此函数的比较对象为时间部分。返回值为布尔值，相等返回 True，否则，返回 False
SameDateTime	DateUtils	此函数可返回两个 TDateTime 变量的比较结果，此函数的比较对象为完整的 TDateTime 变量。返回值为布尔值，相等返回 True，否则，返回 False
IsInLeapYear	DateUtils	此函数可判断指定的 TDateTime 变量是否为闰年，其返回值为布尔值，是闰年则返回 True，否则返回 False
IsLeapYear	DateUtils	此函数可判断指定的年度是否为闰年。其返回值为布尔值，是闰年则返回 True，否则返回 False
IsPM	DateUtils	此函数可判断指定的 TDateTime 变量是否为下午。其返回值为布尔值，是下午则返回 True，否则返回 False
IsSameDay	DateUtils	此函数可判断两个指定的 TDateTime 变量是否为同一天，其比较对象只有 Date。其返回值为布尔值，是同一天则返回 True，否则返回 False
IsToday	DateUtils	此函数可判断指定的 TDateTime 变量是否为今天，其比较对象只有 Date。其返回值为布尔值，是今天则返回 True，否则返回 False
IsValidTime	DateUtils	此函数可判断指定的时间格式是否正确。其返回值为布尔值，格式正确则返回 True，否则返回 False。其中小时数为 0～24；分钟数为 0～59；秒数为 0～59；毫秒数为 0～999
IsValidDate	DateUtils	此函数可判断指定的日期格式是否正确。其返回值为布尔值，格式正确则返回 True，否则返回 False。其中年度为 1～9999；月份为 1～12；日数则视月份而定
IsValidDateTime	DateUtils	此函数可判断指定的日期时间格式是否正确。其返回值为布尔值，格式正确则返回 True，否则返回 False。此函数可视为 IsValidDate 函数与 IsValidTime 函数的结合体
IsValidDateDay	DateUtils	此函数可判断指定的天数是否包含于指定的年度中。其返回值为布尔值，正确则返回 True，否则返回 False

续上表

函数名称	单元文件	所代表的意义
IsValidDateWeek	DateUtils	此函数可判断指定的周数是否包含于指定的年度中。其返回值为布尔值，正确则返回 True，否则返回 False
IsValidDateMonthWeek	DateUtils	此函数可判断指定的日期、月份及周别是否包含于指定的年度中。其返回值为布尔值，是今天则返回 True，否则返回 False。其中年度为 1～9999；月份为 1～12
WithinPastYears	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的年份数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastMonths	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的月份数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastWeeks	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的周数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastDays	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的天数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastHours	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的小时数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastMinutes	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的分钟数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastSeconds	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的秒数范围中。其返回值为布尔值，正确则返回 True，否则返回 False
WithinPastMilli Seconds	DateUtils	此函数可判断指定的 TDateTime 变量是否在指定的毫秒数范围中。其返回值为布尔值，正确则返回 True，否则返回 False

以下将以范例的方式说明每一个函数的使用方法。

CompareDate (返回两个 TDateTime 变量的比较结果)

引用单元：DateUtils

函数声明：function CompareDate(const A, B: TDateTime): TValueRelationship;

范例 D-47

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  DateTime1, DateTime2 : TDateTime;
begin
  DateTime1 := Now;
  DateTime2 := IncMinute(Now,10); //加上 10 分钟
  i := CompareDate(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //0 表示不考虑时间部分
```

```
  DateTime2 := IncMonth(Now,1); //加上1个月
  i := CompareDate(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //-1 表示 DateTime1 < DateTime2
  DateTime2 := IncMonth(Now,-1); //减1个月
  i := CompareDate(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //1 表示 DateTime1 > DateTime2
end;
```

CompareTime (返回两个 TDateTime 变量的比较结果)

引用单元：DateUtils

函数声明：function CompareTime(const A, B: TDateTime): TValueRelationship;

例 D-48

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  DateTime1, DateTime2 : TDateTime;
begin
  DateTime1 := Now;
  DateTime2 := IncMonth(Now,1); //加上1个月
  i := CompareTime(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //0 表示不考虑日期部分
  DateTime2 := IncMinute(Now,10); //加上10分钟
  i := CompareTime(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //-1 表示 DateTime1 < DateTime2
  DateTime2 := IncMinute(Now,-10); //减10分钟
  i := CompareTime(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //1 表示 DateTime1 > DateTime2
end;
```

CompareDateTime (返回两个 TDateTime 变量的比较结果)

引用单元：DateUtils

函数声明：function CompareDateTime(const A, B: TDateTime): TValueRelationship;

例 D-49

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Integer;
  DateTime1, DateTime2 : TDateTime;
begin
```

```
  DateTime1 := Now;
  DateTime2 := IncMonth(Now,1); //加上1个月
  i := CompareDateTime(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //-1 表示 DateTime1 < DateTime2

  DateTime1 := Now; //重新再获取，否则两者会不相同
  DateTime2 := Now;
  i := CompareDateTime(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //0 表示两者变量值相同

  DateTime2 := IncMinute(Now,-10); //减10分钟
  i := CompareDateTime(DateTime1,DateTime2);
  Showmessage(IntToStr(i)); //1 表示 DateTime1 > DateTime2
end;
```

SameDate(返回两个 *TDateTime* 变量的比较结果)

引用单元：DateUtils
函数声明：function SameDate(const A, B: TDateTime): Boolean;

范例 D-50

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
  DateTime1, DateTime2 : TDateTime;
begin
  DateTime1 := Now;
  DateTime2 := IncMinute(Now,10); //加上10分钟
  i := SameDate(DateTime1,DateTime2);
  Showmessage(BoolToStr(i,True)); //True 表示不考虑时间部分

  DateTime2 := IncMonth(Now,1); //加上1个月
  i := SameDate(DateTime1,DateTime2);
Showmessage(BoolToStr(i,True)); //False 表示 DateTime1< DateTime2

  DateTime2 := IncMonth(Now,-1); //减1个月
  i := SameDate(DateTime1,DateTime2);
 Showmessage(BoolToStr(i,True)); //False 表示 DateTime1>DateTime2
end;
```

SameTime(返回两个 *TDateTime* 变量的比较结果)

引用单元：DateUtils
函数声明：function SameTime(const A, B: TDateTime): Boolean;

范例 D-51

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  i : Boolean;
  DateTime1, DateTime2 : TDateTime;
begin
  DateTime1 := Now;
  DateTime2 := IncMonth(Now,1); //加上1个月
  i := SameTime(DateTime1,DateTime2);
  Showmessage(BoolToStr(i,True)); //True 表示不考虑日期部分
  DateTime2 := IncMinute(Now,10); //加上10分钟
  i := SameTime(DateTime1,DateTime2);
Showmessage(BoolToStr(i,True));//False 表示 DateTime1 < DateTime2
  DateTime2 := IncMinute(Now,-10); //减10分钟
  i := SameTime(DateTime1,DateTime2);
Showmessage(BoolToStr(i,True)); //False 表示 DateTime1 > DateTime2
end;
```

※ *SameDateTime*(返回两个 *TDateTime* 变量的比较结果)

引用单元：DateUtils

函数声明：function SameDateTime(const A, B: TDateTime): Boolean;

范例 D-52

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
  DateTime1, DateTime2 : TDateTime;
begin
  DateTime1 := Now;
  DateTime2 := IncMonth(Now,1); //加上1个月
  i := SameDateTime(DateTime1,DateTime2);
  Showmessage(BoolToStr(i,True)); //False 表示 DateTime1 < DateTime2
  DateTime1 := Now; //重新再获取，否则两者会不相同
  DateTime2 := Now;
  i := SameDateTime(DateTime1,DateTime2);
  Showmessage(BoolToStr(i,True)); //True 表示两者变量值相同
  DateTime2 := IncMinute(Now,-10); //减10分钟
  i := SameDateTime(DateTime1,DateTime2);
  Showmessage(BoolToStr(i,True)); //False 表示 DateTime1 > DateTime2
end;
```

- *IsInLeapYear(判断指定的 TDateTime 变量是否为润年)*

引用单元：DateUtils
函数声明：function IsInLeapYear(const AValue: TDateTime): Boolean;

- *IsLeapYear(判断指定的年度是否为润年)*

引用单元：DateUtils
函数声明：function IsLeapYear(Year: Word): Boolean;



D-53

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
begin
  i := IsInLeapYear(Now);
  Showmessage(BoolToStr(i,True)); //True 表示为润年
  i := IsInLeapYear(YearOf(Now));
  Showmessage(BoolToStr(i,True)); //True 表示为润年
end;
```

- *IsPM(判断指定的 TDateTime 变量是否为下午)*

引用单元：DateUtils
函数声明：function IsPM(const AValue: TDateTime): Boolean;

- *IsSameDay(判断两个指定的 TDateTime 变量是否为同一天)*

引用单元：DateUtils
函数声明：function IsSameDay(const AValue, ABasis: TDateTime): Boolean;

- *IsToday(判断指定的 TDateTime 变量是否为今天)*

引用单元：DateUtils
函数声明：function IsToday(const AValue: TDateTime): Boolean;

D-54

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
begin
  i := IsPM(Now);
  Showmessage(BoolToStr(i,True)); //True 表示为下午
```

```
  i := IsSameDay(Now,IncMinute(Now,30)); //相差 30 分钟
  Showmessage(BoolToStr(i,True)); //True 表示同一天
  i := IsToday(IncMinute(Now,30)); //相差 30 分钟
  Showmessage(BoolToStr(i,True)); //True 表示是今天
end;
```

IsValidTime (判断指定的时间格式是否正确)

引用单元：DateUtils
函数声明：function IsValidTime(const AHour, AMinute, ASecond, AMilliSecond: Word): Boolean;



D-55

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
  AHour, AMinute, ASecond, AMilliSecond: Word;
begin
  AHour := 23; //0~24
  AMinute := 0; //0~59
  ASecond := 59; //0~59
  AMilliSecond := 999; //0~999
  i := IsValidTime(AHour, AMinute, ASecond, AMilliSecond);
  Showmessage(BooToStr(i,True)); //True 表示为正确
end;
```

IsValidDate (判断指定的日期格式是否正确)

引用单元：DateUtils
函数声明：function IsValidDate(const AYear, AMonth, ADay: Word): Boolean;



D-56

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
  AYear, AMonth, ADay: Word;
begin
  AYear := 2000; //1~9999
  AMonth := 2; //1~12
  ADay := 29; //视月份而定
  i := IsValidDate(AYear, AMonth, ADay);
  Showmessage(BooToStr(i,True)); //True 表示为正确
end;
```

- *IsValidDateTime*(判断指定的日期时间格式是否正确)

引用单元：DateUtils
函数声明：function IsValidDateTime(const AYear, AMonth, ADay, AHour, AMinute, ASecond, AMilliSecond: Word): Boolean;

范例 D-57

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
  AYear, AMonth, ADay: Word;
  AHour, AMinute, ASecond, AMilliSecond: Word;
begin
  AYear := 2000; //1~9999
  AMonth := 2; //1~12
  ADay := 29; //视月份而定
  AHour := 23; //0~24
  AMinute := 0; //0~59
  ASecond := 59; //0~59
  AMilliSecond := 999; //0~999
  i := IsValidDateTime(AYear, AMonth, ADay, AHour, AMinute, ASecond,
AMilliSecond);
  Showmessage(BoolToStr(i,True)); //True表示为正确
end;
```

- *IsValidDateDay*(判断指定的天数是否包含于指定的年度中)

引用单元：DateUtils
函数声明：function IsValidDateDay(const AYear, ADayOfYear: Word): Boolean;

- *IsValidDateWeek*(判断指定的周数是否包含于指定的年度中)

引用单元：DateUtils
函数声明：function IsValidDateWeek(const AYear, AWeekOfYear, ADayOfWeek: Word): Boolean;

- *IsValidDateMonthWeek*(判断指定的日期、月份及周别是否包含于指定的年度中)

引用单元：DateUtils
函数声明：function IsValidDateMonthWeek(const AYear, AMonth, AWeekOfMonth, ADayOfWeek: Word): Boolean;

范例 D-58

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  i : Boolean;
  AYear, ADayOfYear : Word;
  AWeekOfYear, ADayOfWeek : Word;
  AMonth, AWeekOfMonth : word;
begin
  AYear := 2000; //1~9999
  ADayOfYear := 366; //第366天，会因年度而异
  i := IsValidDateDay(AYear,ADayOfYear);
  Showmessage(BoolToStr(i,True)); //True表示为正确

  AWeekOfYear := 52; //因年度而异
  ADayOfWeek := 1;  //1-7
  i := IsValidDateWeek(AYear, AWeekOfYear, ADayOfWeek);
  Showmessage(BoolToStr(i,True)); //True表示为正确

  AMonth := 3; //1-12
  AWeekOfMonth := 1; //因月份而异
  ADayOfWeek := 3 ; //1-7，因周别而异
  i := IsValidDateMonthWeek(AYear, AMonth, AWeekOfMonth, ADayOfWeek);
  Showmessage(BoolToStr(i,True)); //True表示为正确
end;
```

WithinPastYears(判断指定的 *TDateTime* 变量是否在指定的年份数范围中)

引用单元：DateUtils
函数声明：function WithinPastYears(const ANow, AThen: TDateTime; const AYears: Integer): Boolean;

WithinPastMonths(判断指定的 *TDateTime* 变量是否在指定的月份数范围中)

引用单元：DateUtils
函数声明：function WithinPastMonths(const ANow, AThen: TDateTime; const AMonths: Integer): Boolean;

WithinPastWeeks(判断指定的 *TDateTime* 变量是否在指定的周数范围中)

引用单元：DateUtils
函数声明：function WithinPastWeeks(const ANow, AThen: TDateTime; const AWeeks: Integer): Boolean;

WithinPastDays(判断指定的 *TDateTime* 变量是否在指定的天数范围中)

引用单元：DateUtils
函数声明：function WithinPastDays(const ANow, AThen: TDateTime; const ADays: Integer): Boolean;

- *WithinPastHours*(判断指定的 *TDateTime* 变量是否在指定的小时数范围中)

引用单元：DateUtils
函数声明：function WithinPastHours(const ANow, AThen: TDateTime; const AHours: Int64): Boolean;

- *WithinPastMinutes*(判断指定的 *TDateTime* 变量是否在指定的分钟数范围中)

引用单元：DateUtils
函数声明：function WithinPastMinutes(const ANow, AThen: TDateTime; const AMinutes: Int64): Boolean;

- *WithinPastSeconds*(判断指定的 *TDateTime* 变量是否在指定的秒数范围中)

引用单元：DateUtils
函数声明：function WithinPastSeconds(const ANow, AThen: TDateTime; const ASeconds: Int64): Boolean;

- *WithinPastMilliSeconds*(判断指定的 *TDateTime* 变量是否在指定的毫秒数范围中)

引用单元：DateUtils
函数声明：function WithinPastMilliSeconds(const ANow, AThen: TDateTime; const AMilliSeconds: Int64): Boolean;

范例 D-59

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i : Boolean;
begin
  //第二个日期为过去的第 23 个月，设置比较的条件为 1 年
  i := WithinPastYears(Now,IncMonth(Now,-23),1);
  //如果是过去的第 24 个月，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
  //第二个日期为过去的第 3 个月，设置比较的条件为 3 个月
  i := WithinPastMonths(Now,IncMonth(Now,-3),3);
  //如果是过去的第 4 个月，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
  //第二个日期为过去的第 3 周，设置比较的条件为 3 周
  i := WithinPastWeeks(Now,IncWeek(Now,-3),3);
  //如果是过去的第 4 周，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
  //第二个日期为过去的第 3 小时，设置比较的条件为 3 小时
```

```
  i := WithinPastHours(Now,IncHour(Now,-3),3);
  //如果是过去的第 4 小时，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
  //第二个日期为过去的第 3 分钟，设置比较的条件为 3 分钟
  i := WithinPastMinutes(Now,IncMinute(Now,-3),3);
  //如果是过去的第 4 分钟，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
  //第二个日期为过去的第 3 秒，设置比较的条件为 3 秒
  i := WithinPastSeconds(Now,IncSecond(Now,-3),3);
  //如果是过去的第 4 秒，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
  //第二个日期为过去的第 3 毫秒，设置比较的条件为 3 毫秒
  i := WithinPastMilliSeconds(Now,IncMilliSecond(Now,-3),3);
  //如果是过去的第 4 秒，则返回 False
  Showmessage(BoolToStr(i,True)); //True 表示为正确
end;
```

D.5　日期与时间的编译与解译

本小节将为您介绍 Delphi 6 所提供日期与时间的编译与解译函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 D.5 所示。

表 D.5

函数名称	单元文件	所代表的意义
DecodeDate	SysUtils	此函数可将指定的 TdateTime 变量分解为年度、月份及日期
DecodeTime	SysUtils	此函数可将指定的 TdateTime 变量分解为时、分、秒及毫秒
DecodeDateTime	DateUtils	此函数可将指定的 TdateTime 变量分解为年度、月份、日期、时、分、秒及毫秒
DecodeDateDay	DateUtils	此函数可将指定的 TdateTime 变量分解为年度及年度的第几天
DecodeDateWeek	DateUtils	此函数可将指定的 TdateTime 变量分解为年度、周别及该周的第几天
DecodeDateFully	SysUtils	此函数可将指定的 TdateTime 变量分解为年度、月份、日期及该周的第几天
DecodeDateMonthWeek	DateUtils	此函数可将指定的 TdateTime 变量分解为年度、月份、该月的第几周及该周的第几天
DecodeDayOfWeekInMonth	DateUtils	此函数可将指定的 TdateTime 变量分解为年度、月份、该月的第几周及该周的第几天

续上表

函数名称	单元文件	所代表的意义
EncodeDate	SysUtils	此函数可将指定的年、月、日参数转换 TDateTime 类型的日期格式
TryEncodeDate	SysUtils	此函数除了与 EncodeDate 函数具有相同的功能外，另可判断指定的参数是否正确。当返回值为 True 表示正确，反之，则返回 False
EncodeTime	SysUtils	此函数可将指定的时、分、秒、毫秒等参数转换 TDateTime 类型的时间格式
TryEncodeTime	SysUtils	此函数除了与 EncodeTime 函数具有相同的功能外，另可判断指定的参数是否正确。当返回值为 True 表示正确，反之，则返回 False
EncodeDateTime	SysUtils	此函数可将指定的年、月、日、时、分、秒、毫秒等参数转换 TDateTime 类型的日期时间格式
TryEncodeDate Time	SysUtils	此函数除了与 EncodeDateTime 函数具有相同的功能外，另可判断指定的参数是否正确。当返回值为 True 表示正确，反之，则返回 False

以下将以范例的方式说明每一个函数的使用方法。

- *DecodeDate (将指定的 TDateTime 变量分解为年度、月份及日期)*

引用单元：SysUtils
函数声明：procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);

- *DecodeTime (将指定的 TDateTime 变量分解为时、分、秒及毫秒)*

引用单元：SysUtils
函数声明：procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);

- *DecodeDateTime(将指定的 TDateTime 变量分解为年度、月份、日期、时、分、秒及毫秒)*

引用单元：DateUtils
函数声明：procedure DecodeDateTime(const AValue: TDateTime; out AYear, AMonth, ADay, AHour, AMinute, ASecond, AMilliSecond: Word);

范例 **D-60**

```
procedure TForm1.Button1Click(Sender: TObject);
var
    AValue: TDateTime;
```

```
  AYear, AMonth, ADay, AHour, AMinute,ASecond, AMilliSecond: Word;
  Str : String;
begin
  AValue := Now;
  DecodeDate(AValue,AYear, AMonth, ADay);
  Str := '今天是' + IntToStr(AYear) + '年' + IntToStr(AMonth) + '月' +
       IntToStr(ADay) + '日';
  Showmessage(Str);
  DecodeTime(AValue, AHour, AMinute, ASecond, AMilliSecond);
  Str := '现在是' + IntToStr(AHour) + '时' + IntToStr(AMinute) + '分' +
       IntToStr(ASecond)  '秒' + IntToStr(AMilliSecond) + '毫秒';
  Showmessage(Str);
  DecodeDateTime(AValue,AYear, AMonth, ADay, AHour, AMinute,
                      ASecond, AMilliSecond);
  Str :  '现在是' + IntToStr(AYear) + '年' + IntToStr(AMonth) + '月' +
         IntToStr(ADay) + '日' + IntToStr(AHour) + '时' +
         IntToStr(AMinute) + '分' + IntToStr(ASecond) + '秒' +
         IntToStr(AMilliSecond) + '毫秒';
  Showmessage(Str);
end;
```

- *DecodeDateDay(将指定的 TDateTime 变量分解为年度及年度的第几天)*

引用单元：SysUtils
函数声明：procedure DecodeDateDay(const AValue: TDateTime; out AYear, ADayOfYear: Word);

- *DecodeDateWeek(将指定的 TDateTime 变量分解为年度、周别及该周的第几天)*

引用单元：SysUtils
函数声明：procedure DecodeDateWeek(const AValue: TDateTime; out AYear, AWeekOfYear, ADayOfWeek: Word);

- *DecodeDateFully(将指定的 TDateTime 变量分解为年度、月份、日期及该周的第几天)*

引用单元：SysUtils
函数声明：function DecodeDateFully(const DateTime: TDateTime; var Year, Month, Day, ADayOfWeek: Word): Boolean;

- *DecodeDateMonthWeek(将指定的 TDateTime 变量分解为年度、月份、该月的第几周、及该周的第几天)*

引用单元：SysUtils
函数声明：procedure DecodeDateMonthWeek(const AValue: TDateTime; out AYear, AMonth, AWeekOfMonth, ADayOfWeek: Word);

DecodeDayOfWeekInMonth(将指定的 *TDateTime* 变量分解为年度、月份、及该周的第几天)

引用单元：SysUtils

函数声明：procedure DecodeDayOfWeekInMonth(const AValue: TDateTime; out AYear, AMonth, ANthDayOfWeek, ADayOfWeek: Word);

范例 D-61

```
procedure TForm1.Button1Click(Sender: TObject);
var
   AValue: TDateTime;
   AYear, AMonth, ADay, ADayOfYear, AWeekOfMonth,
   AWeekOfYear, ADayOfWeek,ANthDayOfWeek : word;
   Str : String;
begin
   AValue := Now;
   DecodeDateDay(AValue,AYear,ADayOfYear);
   Str:='今天是'+ IntToStr(AYear) + '年的第' + IntToStr(ADayOfYear)
            + '天';
   Showmessage(Str);

   DecodeDateWeek(AValue, AYear, AWeekOfYear, ADayOfWeek);
   Str:= '今天是' + IntToStr(AYear) + '年. 第' + IntToStr(AWeekOfYear)
            + '周的第' + IntToStr(ADayOfWeek) + '天';
   Showmessage(Str);

   DecodeDateFully(AValue, AYear,AMonth, ADay, ADayOfWeek);
   Str := '今天是' + IntToStr(AYear) + '年' + IntToStr(AMonth) + '月' +
    IntToStr(ADay) +'日，且为该周的第'+IntToStr(ADayOfWeek)+'天';
   Showmessage(Str);

   DecodeDateMonthWeek(AValue, AYear, AMonth, AWeekOfMonth,
                                        ADayOfWeek);
   Str := '今天是' + IntToStr(AYear) + '年' + IntToStr(AMonth) + '月' +
     IntToStr(ADay) +'日，且为该月的第' + IntToStr(AWeekOfMonth)
     + '周，且为该周的第' + IntToStr(ADayOfWeek) + '天';
   Showmessage(Str);

   DecodeDayOfWeekInMonth(AValue, AYear, AMonth,
                                   AnthDayOfWeek , ADayOfWeek);
   Str := '今天是' + IntToStr(AYear) + '年' + IntToStr(AMonth) + '月' +
   IntToStr(ADay) +'日，且为该月的第' + IntToStr(ANthDayOfWeek) +
   '周(ANthDayOfWeek)与该周的第' + IntToStr(ADayOfWeek) +
   '天(ADayOfWeek)';
   Showmessage(Str);
end;
```

EncodeDate(将指定的年、月、日参数转换 *TDateTime* 类型的日期格式)

引用单元：SysUtils

函数声明：function EncodeDate(Year, Month, Day: Word): TDateTime;

范例 D-62

```
var
  TempDateTime : TDateTime;
  AYear, AMonth, ADay : word;
begin
  AYear := 2001;
  AMonth := 2;
  ADay := 29;
  try
    TempDateTime := EncodeDate(AYear, AMonth, ADay);
    Showmessage(DateToStr(TempDateTime));
  except
    //当参数错误则触发一个异常
    on E: Exception do Showmessage(E.Message);
  end;
end;
```

TryEncodeDate(将指定的年、月、日参数转换 *TDateTime* 类型的日期格式)

引用单元：SysUtils

函数声明：function TryEncodeDate(Year, Month, Day: Word; out Date: TDateTime): Boolean;

范例 D-63

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TempDateTime : TDateTime;
  AYear, AMonth, ADay : word;
begin
  AYear := 2001;
  AMonth := 2;
  ADay := 29;
  if TryEncodeDate(AYear, AMonth, ADay,TempDateTime) then
    Showmessage(DateToStr(TempDateTime))
  else
    //当参数不正确时 TryEncodeDate 函数返回 False
    Showmessage('指定的参数不正确');
end;
```

EncodeTime(将指定的时、分、秒、毫秒等参数转换 *TDateTime* 类型的时间格式)

引用单元：SysUtils

函数声明：function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime;

例 D-64

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TempDateTime : TDateTime;
   Hour, Min, Sec, MSec : word;
begin
   Hour := 24; //0-23
   Min := 3;
   Sec := 29;
   MSec := 888;
   try
      TempDateTime := EncodeTime(Hour, Min, Sec, MSec);
      Showmessage(TimeToStr(TempDateTime));
   except
      //当参数错误则触发一个异常
      on E: Exception do Showmessage(E.Message);
   end;
end;
```

TryEncodeTime(将指定的时、分、秒、毫秒等参数转换 *TDateTime* 类型的时间格式)

引用单元：SysUtils

函数声明：function TryEncodeTime(Hour, Min, Sec, MSec: Word; out Time: TDateTime): Boolean;

例 D-65

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TempDateTime : TDateTime;
   Hour, Min, Sec, MSec : word;
begin
   Hour := 24; //0-23
   Min := 3;
   Sec := 29;
   MSec := 888;
   if TryEncodeTime(Hour, Min, Sec, MSec,TempDateTime) then
      Showmessage(TimeToStr(TempDateTime))
```

```
  else
    //当参数不正确时 TryEncodeDate 函数返回 False
    Showmessage('指定的参数不正确');
end;
```

- *EncodeDateTime*(将指定的年、月、日、时、分、秒、毫秒等参数转换 *TdateTime* 类型的日期时间格式)

引用单元：SysUtils
函数声明：function EncodeDateTime(const AYear, AMonth, ADay, AHour, AMinute, ASecond, AMilliSecond: Word):TDateTime;

- *TryEncodeDateTime*(将指定的年、月、日、时、分、秒、毫秒等参数转换 *TdateTime* 类型的日期时间格式)

引用单元：SysUtils
函数声明：function TryEncodeDateTime(const AYear, AMonth, ADay, AHour, AMinute, ASecond, AMilliSecond: Word; out AValue: TDateTime): Boolean;

范例 D-66

```
procedure TForm1.button1Click(Sender: TObject);
var
  TempDateTime : TDateTime;
  AYear, AMonth, ADay, Hour, Min, Sec, MSec : word;
begin
  AYear := 2001;
  AMonth := 2;
  ADay := 29;  //2001年2月只有28天
  Hour := 24; //0-23
  Min := 3;
  Sec := 29;
  MSec := 888;
  try
    TempDateTime := EncodeDateTime(AYear, AMonth, ADay,Hour, Min, Sec,
MSec);
    Showmessage(TimeToStr(TempDateTime));
  except
    //当参数错误则触发一个异常
    on E: Exception do Showmessage(E.Message);
  end;
  if TryEncodeDateTime(AYear, AMonth, ADay,Hour, Min, Sec,
                              MSec,TempDateTime) then
    Showmessage(TimeToStr(TempDateTime))
```

```
  else
    Showmessage('参数错误');
end;
```

D.6 日期与时间的转换函数

本小节将为您介绍 Delphi 6 所提供日期与时间的转换函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 D.6 所示。

表 D.6

函数名称	单元文件	所代表的意义
DateTimeToFile Date	SysUtils	此函数可以将 TDateTime 的日期格式转换为操作系统时间(OS timestamp)格式，以供 FileSetDate 函数设置文件的时间
TimeToStr	SysUtils	此函数可以将 TDateTime 的时间转为字符串
DateToStr	SysUtils	此函数可以将 TDateTime 的日期转为字符串
DateTimeToStr	SysUtils	此函数可以将 TDateTime 的日期时间转为字符串
DateTimeToString	SysUtils	此函数可将日期时间转为指定格式的字符串
FormatDateTime	SysUtils	此函数可按指定格式将 TDateTime 变量转为字符串
StrToDate	SysUtils	此函数可将指定的字符串转为 TDateTime 的日期格式
StrToTime	SysUtils	此函数可将指定的字符串转为 TDateTime 的时间格式
StrToDateTime	SysUtils	此函数可将指定的字符串转为 TDateTime 的日期时间格式
ReplaceDate	SysUtils	此函数可使用新 TDateTime 变量中的日期替换原有 TDateTime 变量的日期部分
ReplaceTime	SysUtils	此函数可使用新 TDateTime 变量中的时间替换原有 TDateTime 变量的时间部分
TimeOf	DateUtils	此函数会将指定 TDateTime 变量的日期改为 1899-12-30 但保留时间部分
DateOf	DateUtils	此函数会将指定 TDateTime 变量的时间改为午夜 AM 00:00:00 但保留日期部分

以下将以范例的方式说明每一个函数的使用方法。

DateTimeToFileDate(将 TDateTime 的日期格式转换为操作系统时间格式)

引用单元：SysUtils
函数声明：function DateTimeToFileDate(DateTime: TDateTime): Integer;

TimeToStr(将时间转为字符串)

引用单元：SysUtils
函数声明：function TimeToStr(Time: TDateTime): string;

- *DateToStr(将日期转为字符串)*

引用单元：SysUtils
函数声明：function DateToStr(Date: TDateTime): string;

- *DateTimeToStr(将日期时间转为字符串)*

引用单元：SysUtils
函数声明：function DateTimeToStr(DateTime: TDateTime): string;

- *DateTimeToString(将日期时间转为指定格式的字符串)*

引用单元：SysUtils
函数声明：procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime);

- *FormatDateTime(将日期时间转为指定格式的字符串)*

引用单元：SysUtils
函数声明：function FormatDateTime(const Format: string; DateTime: TDateTime): string;

范例 D-67

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TempDateTime : TDateTime;
  Temp : Integer;
  Str : String;
begin
  //DateTimeToFileDate 函数
  TempDateTime := Now;
  Temp := DateTimeToFileDate(TempDateTime);
  //TimeToStr 函数
  Showmessage(TimeToStr(TempDateTime));
  //DateToStr 函数
  Showmessage(DateToStr(TempDateTime));
  //DateTimeToStr 函数
  Showmessage(DateTimeToStr(TempDateTime));
  //DateTimeToString 函数
  DateTimeToString(Str,'yyyy,mmmm d,dddd
                        hh:mm' ,TempDateTime);
  Showmessage(Str);
  //FormatDateTime 函数
  Str := FormatDateTime('"现在是"yyyy,mmmm d,dddd
```

```
                                        hh:mm',TempDateTime);
  Showmessage(Str);
end;
```

- *StrToDate(将字符串转为 TDateTime 的日期格式)*

引用单元：SysUtils
函数声明：function StrToDate(const S: string): TDateTime;

- *StrToTime(将字符串转为 TDateTime 的时间格式)*

引用单元：SysUtils
函数声明：function StrToTime(const S: string): TDateTime;

- *StrToDateTime(将字符串转为 TDateTime 的日期时间格式)*

引用单元：SysUtils
函数声明：function StrToDateTime(const S: string): TDateTime;

范例 D-68

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TempDateTime : TDateTime;
   Str : String;
begin
   //StrToDate 函数
   Str := '2001/01/01';
   TempDateTime := StrToDate(Str);
   Showmessage(DateToStr(TempDateTime));
   //StrToTime 函数
   Str := '23:01:23';
   TempDateTime := StrToTime(Str);
   Showmessage(TimeToStr(TempDateTime));
   //StrToDateTime 函数
   Str := '2001/01/01 23:01:23';
   TempDateTime := StrToDateTime(Str);
   Showmessage(DateTimeToStr(TempDateTime));
end;
```

- *ReplaceDate(使用新 TDateTime 变量中的日期替换原有 TDateTime 变量的日期部分)*

引用单元：SysUtils
函数声明：procedure ReplaceDate(var DateTime: TDateTime; const NewDate: TDateTime);

- *ReplaceTime(使用新 TDateTime 变量中的时间替换原有 TDateTime 变量的时间部分)*

引用单元：SysUtils
函数声明：procedure ReplaceTime(var DateTime: TDateTime; const NewTime: TDateTime);



D-69

```
procedure TForm1.Button1Click(Sender: TObject);
var
  OldDateTime,NewDateTime : TDateTime;
  Str : String;
begin
  //ReplaceDate 函数
  OldDateTime := Now;
  NewDateTime := IncMonth(Now,2);
  Str := '转换前:' + DateTimeToStr(OldDateTime) + #10;
  Str := Str + '新日期:' + DateTimeToStr(NewDateTime) + #10;
  ReplaceDate(OldDateTime,NewDateTime);
  Str := Str + '转换后:' + DateTimeToStr(OldDateTime) + #10;
  Showmessage(Str);
  //ReplaceTime 函数
  OldDateTime := Now;
  NewDateTime := IncSecond(Now,20);
  Str := '转换前:' + DateTimeToStr(OldDateTime) + #10;
  Str := Str + '新日期:' + DateTimeToStr(NewDateTime) + #10;
  ReplaceTime(OldDateTime,NewDateTime);
  Str := Str + '转换后:' + DateTimeToStr(OldDateTime) + #10;
  Showmessage(Str);
end;
```

- *TimeOf(将指定 TDateTime 变量的日期改为 1899-12-30 但保留时间部分)*

引用单元：DateUtils
函数声明：function TimeOf(const AValue: TDateTime): TDateTime;

- *DateOf(将指定 TDateTime 变量的时间改为午夜 AM 00:00:00 但保留日期部分)*

引用单元：DateUtils
函数声明：function DateOf(const AValue: TDateTime): TDateTime;



D-70

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   TempDateTime : TDateTime;
   Str : String;
begin
   //TimeOf 函数
   TempDateTime := Now;
   Str := '原日期时间=' + DateTimeToStr(TempDateTime) + #10;
   TempDateTime := TimeOf(TempDateTime);
   Str := Str + '新日期时间=' + DateTimeToStr(TempDateTime) + #10;
   Showmessage(Str);
   //DateOf 函数
   TempDateTime := Now;
   Str := '原日期时间=' + DateTimeToStr(TempDateTime) + #10;
   TempDateTime := DateOf(TempDateTime);
   Str := Str + '新日期时间=' + DateTimeToStr(TempDateTime) + #10;
   Showmessage(Str);
end;
```

附 录 E

Delphi 6 其他常用函数

本 章 目 标

每一种程序语言都会提供一些内建函数供程序设计师使用，这些内建函数可以节省程序设计师很多时间，当然 Delphi 6 也不异常。前面两个章节中，笔者已经介绍了文件处理函数及日期时间处理函数。在本章中，笔者要继续为您介绍 Delphi 6 其他的内建函数。本章所要介绍的范围包括字符串、数值及 Variant 等函数。因为 De_phi 6 的字符串可分为 Pascal 字符串与以 Null 结尾的字符串，因此我们将分别为您说明 Pascal 字符串处理函数及以 Null 结尾的字符串处理函数等。

本 章 主 题

E.1 Pascal 字符串处理函数

E.2 以 Null 结尾的字符串处理函数

E.3 数值处理函数

E.4 Variant 处理函数

E.1 Pascal 字符串处理函数

本小节笔者将为您介绍 Delphi 6 所提供的 Pascal 字符串处理函数。那什么是 Pascal 字符串呢？所谓 Pascal 字符串代表的是一连串的字符，Delphi 6 所默认的字符串类型包括 ShortString、AnsiString 及 WideString 等。其中 ShortString 就是所谓的短字符串，其长度为 0 到 255 个字符。AnsiString 我们称其为长字符串，代表的是一种动态记忆分配的字符串，其字符串内容使用 8 位的 ANSI 字符来表示。理论上，它并无最大长度的限制，但实务上还是受限于可使用的内存空间。WideString 也是一种动态分配内存的字符串，与 AnsiString 的不同之处在于 WideString 的字符串内容使用 16 位的 Unicode 字符来表示。有关 Pascal 字符串的详细介绍请参考附录 A 的数据类型章节。

本小节将为您介绍 Delphi 6 所提供的 Pascal 字符串处理函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个 Pascal 字符串处理函数所代表的意义，如表 E.1 所示。

表 E.1

函数名称	单元文件	所代表的意义
LowerCase	SysUtils	此函数可以指定字符串中的所有字符转为小写字母。但其只针对 7 位的英文字母
UpperCase	SysUtils	此函数可以指定字符串中的所有字符转为大写字母。但其只针对 7 位的英文字母
AnsiLowerCase	SysUtils	此函数可以指定字符串中的所有字符转为小写字母。支持 multi-byte character sets (MBCS)的英文字母
AnsiUpperCase	SysUtils	此函数可以指定字符串中的所有字符转为大写字母。支持 multi-byte character sets (MBCS)的英文字母
WideLowerCase	SysUtils	此函数可以指定的 Unicode 的所有字符转为小写字母
WideUpperCase	SysUtils	此函数可以指定的 Unicode 的所有字符转为大写字母
Trim	SysUtils	此函数可以指定字符串前后的空格符加以消除，但无法清除全角空格
TrimLeft	SysUtils	此函数可以指定字符串前面(也就是左边)的空格符加以消除，但无法清除全角空格
TrimRight	SysUtils	此函数可以指定字符串后面(也就是右边)的空格符加以消除，但无法清除全角空格
CompareStr	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为不同。当两个字符串相等时返回 0，否则返回非 0 的整数
CompareText	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为相同。当两个字符串相等时返回 0，否则返回非 0 的整数

续上表

函数名称	单元文件	所代表的意义
AnsiCompareStr	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为不同。当两个字符串相等时返回 0，否则返回非 0 的整数
AnsiCompareText	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为相同。当两个字符串相等时返回 0，否则返回非 0 的整数
SameText	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为相同。当两个字符串相等时返回 True，否则返回 False
AnsiSameStr	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为不同。当两个字符串相等时返回 True，否则返回 False
AnsiSameText	SysUtils	此函数可以比较两字符串的字母大小，比较过程中字母的大小写被视为相同。当两个字符串相等时返回 True，否则返回 False
WideSameStr	SysUtils	此函数可以比较两个 Unicode 字符串的字母大小，比较过程中字母的大小写被视为相同。当两个字符串相等时返回 True，否则返回 False
WideSameText	SysUtils	此函数可以比较两个 Unicode 字符串的字母大小，比较过程中字母的大小写被视为相同。当两个字符串相等时返回 True，否则返回 False
AnsiMatchStr	StrUtils	此函数可以判断子字符串是否存在于指定的字符串数组中，比较过程中字母的大小写被视为不同
AnsiMatchText	StrUtils	此函数可以判断子字符串是否存在于指定的字符串数组中，比较过程中字母的大小写被视为相同
AnsiContainsStr	StrUtils	此函数可以判断子字符串是否存在于指定的字符串中，比较过程中字母的大小写被视为不同
AnsiContainsText	StrUtils	此函数可以判断子字符串是否存在于指定的字符串中，比较过程中字母的大小写被视为相同
AnsiReplaceStr	StrUtils	此函数可以在指定字符串中替换指定的子字符串，在查找的过程中字母的大小写被视为不同
AnsiReplaceText	StrUtils	此函数可以在指定字符串中替换指定的子字符串，在查找的过程中字母的大小写被视为相同
MidStr	StrUtils	此函数可从指定位置获取指定数量的子字符串
LeftStr	StrUtils	此函数可从字符串的左边位置获取指定数量的子字符串
RightStr	StrUtils	此函数可从字符串的右边位置获取指定数量的子字符串

续上表

函数名称	单元文件	所代表的意义
Pos	System	此函数可查找子字符串在字符串中的位置
AnsiPos	SysUtils	此函数与 Pos 函数相同可以找出子字符串在指定字符串中的位置。假如找到则返回子字符串的位置，否则返回 0。此函数支持 multi-byte character sets (MBCS)
Length	System	此函数可以返回指定字符串的长度，其返回值为整数值
Copy	System	此函数可以从指定位置获取指定数量的子字符串，其用法与 MidStr 相似
Delete	System	此函数可以从指定位置删除指定数量的子字符串
Concat	System	此函数可将指定的字符串相加成一个新的字符串，不限字符串个数
Insert	System	此函数可以指定子字符串插入另一个字符串的指定位置
AnsiStartsStr	StrUtils	此函数可判断子字符串是否为母字符串的开头，当返回值为 True 表示是开头字符串，反之，则不是。比较过程，字母大小写视为不同
AnsiStartsText	StrUtils	此函数可判断子字符串是否为母字符串的开头，当返回值为 True 表示是开头字符串；反之，则不是。比较过程，字母大小写视为相同
AnsiEndsStr	StrUtils	此函数可判断子字符串是否为母字符串的结尾，当返回值为 True 表示是开头字符串，反之，则不是。比较过程，字母大小写视为不同
AnsiEndsText	StrUtils	此函数可判断子字符串是否为母字符串的结尾，当返回值为 True 表示是开头字符串，反之，则不是。比较过程，字母大小写视为相同
AnsiIndexStr	StrUtils	此函数可返回指定字符串为字符串数组中的位置，比较过程中，字母大小写视为不同
AnsiIndexText	StrUtils	此函数可返回指定字符串为字符串数组中的位置，比较过程中，字母大小写视为相同
QuotedStr	SysUtils	此函数可为指定字符串加上一组单引号(')
AnsiQuotedStr	SysUtils	此函数可为指定字符串加上一组指定字符的符号于字符串的前后
AnsiDequotedStr	SysUtils	此函数为 AnsiQuotedStr 函数的反向，也就是可以去除字符串前后的指定字符
IsDelimiter	SysUtils	此函数可判断指定的子字符串是否出现在指定的位置。其比较时将字母大小写视为不同，而且只以子字符串的第一个字母进行比较

续上表

函数名称	单元文件	所代表的意义
LastDelimiter	SysUtils	此函数可判断指定的子字符串中任何一个字符出现在指定字符串中的最后一个位置，其比较时将字母大小写视为不同
Str	System	此函数可将数值转换为指定格式的字符串
Val	System	此函数可将指定的字符串转为数字，而且可以得知发生转换错误的字符为何
StringOfChar	System	此函数可返回指定个数的字符，其返回值为字符串类型
StuffString	StrUtils	此函数可在指定位置与长度插入指定的子字符串
WrapText	SysUtils	此函数可将指定字符串依照指定长度加以分隔成多行的字符串
SetLength	System	此函数可动态更改动态字符串的长度，其长度必须介于 0 到 255
DupeString	StrUtils	此函数与 StringOfChar 函数在功能上相似，其最大的不同在于前者重复的对象是字符串，而后者是字符
ReverseString	StrUtils	此函数可将指定的字符串加以反转为新的字符串

以下将以范例的方式说明每一个函数的使用方法。

- *LowerCase(将所有字符转为小写字母)*

引用单元：SysUtils

函数声明：function LowerCase(const S: string): string;

范例 E-1

```
procedure TForm1.Button1Click(Sender: TObject);
var
   MyString:String;
begin
   MyString:='DELPHI IS GOOD!!';
   ShowMessage(LowerCase(MyString));
end;
```

- *UpperCase(将所有字符转为大写字母)*

引用单元：SysUtils

函数声明：function UpperCase(const S: string): string;

范例 E-2

```
procedure TForm1.Button1Click(Sender: TObject);
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   MyString:String;
begin
   MyString:='Delphi is good!!';
   ShowMessage(UpperCase(MyString));
end;
```

- *AnsiLowerCase(将所有字符转为小写字母)*

引用单元：SysUtils
函数声明：function AnsiLowerCase(const S: string): string;



E-3

```
procedure TForm1.Button1Click(Sender: TObject);
var
   MyString:String;
begin
   MyString:='DELPHI IS GOOD!!';
   ShowMessage(AnsiLowerCase(MyString));
end;
```

- *AnsiUpperCase(将所有字符转为大写字母)*

引用单元：SysUtils
函数声明：function AnsiUpperCase(const S: string): string;



E-4

```
procedure TForm1.Button1Click(Sender: TObject);
var
   MyString:String;
begin
   MyString:='Delphi is good!!';
   ShowMessage(AnsiUpperCase(MyString));
end;
```

- *WideLowerCase(将所有 Unicode 字符转为小写字母)*

引用单元：SysUtils
函数声明：function AnsiLowerCase(const S: string): string;



E-5

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    MyString:WideString;
begin
    MyString:='DELPHI IS GOOD!!';
    ShowMessage(WideLowerCase(MyString));
end;
```

- *WideUpperCase(将所有 Unicode 字符转为大写字母)*

引用单元：SysUtils
函数声明：function WideUpperCase(const S: WideString): WideString;



E-6

```
procedure TForm1.Button1Click(Sender: TObject);
var
    MyString : WideString;
begin
    MyString:='Delphi is good!!';
    ShowMessage(WideUpperCase(MyString));
end;
```

- *Trim(清除字符串前后的空格)*

引用单元：SysUtils
函数声明：function Trim(const S: string): string; function Trim(const S: WideString): WideString;

- *TrimLeft(清除字符串左边的空格)*

引用单元：SysUtils
函数声明：function TrimLeft(const S: string): string; function TrimLeft(const S: WideString): WideString;

- *TrimRight(清除字符串右边的空格)*

引用单元：SysUtils
函数声明：function TrimRight(const S: string): string; function TrimRight(const S: WideString): WideString;



E-7

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1:String;
begin
   //TrimLeft,TrimRight,Trim 无法处理全角空格
```

```
  S1:='  delphi  '; //前后为半角空格符
  ShowMessage(''''+TrimLeft(S1)+''''');    {'delphi  '}
  ShowMessage(''''+TrimRight(S1)+''''');   {'  delphi'}
  ShowMessage(''''+Trim(S1)+''''');        {'delphi'}
  ShowMessage(''''+S1+''''');              {'  delphi  '}
end;
```

CompareStr(比较两字符串的大小,大小写字母视为不同)

引用单元：SysUtils
函数声明：function CompareStr(const S1, S2: string): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

范例 E-8

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S1,S2:String;
begin
  S1:='Delphi';
  S2:='delphi';
  if CompareStr(S1,S2)<>0 then
    ShowMessage('大小写被视为不同喔!!');
end;
```

CompareText (比较两字符串的大小,大小写字母视为相同)

引用单元：SysUtils
函数声明：function CompareText(const S1, S2: string): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

范例 E-9

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S1,S2:String;
begin
  S1:='Delphi';
  S2:='delphi';
  if CompareText(S1,S2)=0 then
```

```
    ShowMessage('大小写被视为相同喔!!');
end;
```

- *AnsiCompareStr(比较两字符串的大小,大小写字母视为不同)*

引用单元：SysUtils
函数声明：function AnsiCompareStr(const S1, S2: string): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0



E-10

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2:String;
begin
   S1:='Delphi';
   S2:='delphi';
   if AnsiCompareStr (S1,S2)<>0 then
      ShowMessage('大小写被视为不同喔!!');
end;
```

- *AnsiCompareText(比较两字符串的大小,大小写字母视为相同)*

引用单元：SysUtils
函数声明：function AnsiCompareText(const S1, S2: string): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0



E-11

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2:String;
begin
   S1:='Delphi';
   S2:='delphi';
   if AnsiCompareText (S1,S2)=0 then
      ShowMessage('大小写被视为相同喔!!');
end;
```

SameText(比较两字符串的大小,大小写字母视为相同)

引用单元：SysUtils
函数声明：function SameText(const S1, S2: string): Boolean;
返回结果：当 S1 = S2 返回 True，否则返回 False

范例 E-12

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S1,S2:String;
begin
  S1:='Delphi';
  S2:='delphi';
  if SameText(S1,S2) then
    ShowMessage('大小写被视为相同喔!!');
end;
```

AnsiSameStr(比较两字符串的大小,大小写字母视为不同)

引用单元：SysUtils
函数声明：function AnsiSameStr(const S1, S2: string): Boolean;
返回结果：当 S1 = S2 返回 True，否则返回 False

范例 E-13

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S1,S2:String;
begin
  S1:='Delphi';
  S2:='delphi';
  if NOT AnsiSameStr(S1,S2) then
    ShowMessage('大小写被视为不同喔!!');
end;
```

AnsiSameText(比较两字符串的大小,大小写字母视为相同)

引用单元：SysUtils
函数声明：function AnsiSameText(const S1, S2: string): Boolean;
返回结果：当 S1 = S2 返回 True，否则返回 False

范例 E-14

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   S1,S2:String;
begin
   S1:='Delphi';
   S2:='delphi';
   if AnsiSameText(S1,S2) then
      ShowMessage('大小写被视为相同哦!!');
End;
```

- *WideSameStr(比较两个 Unicode 字符串的大小,大小写字母视为不同)*

引用单元：SysUtils

函数声明：function WideSameStr(const S1, S2: WideString): Boolean;

返回结果：当 S1 = S2 返回 True，否则返回 False

范例 E-15

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2:WideString;
begin
   S1:='Delphi';
   S2:='delphi';
   if NOT WideSameStr(S1,S2) then
      ShowMessage('大小写被视为不同哦!!');
End;
```

- *WideSameText(比较两字符串的大小,大小写字母视为相同)*

引用单元：SysUtils

函数声明：function WideSameText(const S1, S2: WideString): Boolean;

返回结果：当 S1 = S2 返回 True，否则返回 False

范例 E-16

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2:WideString;
begin
   S1:='Delphi';
   S2:='delphi';
   if WideSameText(S1,S2) then
      ShowMessage('大小写被视为相同哦!!');
End;
```

- *AnsiMatchStr*(判断子字符串是否存在于指定的字符串数组中，大小写字母视为不同)

引用单元：StrUtils
函数声明：function AnsiMatchStr(const AText: string; const AValues: array of string): Boolean;

范例 E-17

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2,S3,S4:String;
begin
   //uses StrUtils
   S1:='Delphi';
   S2:='Test';
   S3:='Delphi';
   S4:='Nothing';
   if AnsiMatchStr(S1,[S2,S3,S4]) then
      ShowMessage('大小写被视为不同喔!!');
End;
```

- *AnsiMatchText*(判断子字符串是否存在于指定的字符串数组中，大小写字母视为相同)

引用单元：StrUtils
函数声明：function AnsiMatchText(const AText: string; const AValues: array of string): Boolean;

范例 E-18

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2,S2,S4:String;
begin
   //uses StrUtils
   S1:='Delphi';
   S2:='Test';
   S3:='delphi';
   S4:='Nothing';
   if AnsiMatchText(S1,[S2,S3,S4]) then
      ShowMessage('大小写被视为相同喔!!');
End;
```

- *AnsiContainsStr(判断子字符串是否存在于指定的字符串中，大小写字母视为不同)*

引用单元：StrUtils

函数声明：function AnsiContainsStr(const AText, ASubText: string): Boolean;



```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2:String;
begin
   //uses StrUtils
   S1:='Delphi'; //子字符串
   S2:='I love the Delphi tool'; //主要字符串
   if AnsiContainsStr(S2, S1) then
     ShowMessage('大小写被视为不同喔!!');
End;
```

- *AnsiContainsStr(判断子字符串是否存在于指定的字符串中，大小写字母视为相同)*

引用单元：StrUtils

函数声明：function AnsiContainsText(const AText, ASubText: string): Boolean;



```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2:String;
begin
   //uses StrUtils
   S1:='Delphi'; //子字符串
   S2:='I love the delphi tool'; //主要字符串
   if AnsiContainsText(S2, S1) then
     ShowMessage('大小写被视为相同喔!!');
End;
```

- *AnsiReplaceStr(置换指定的子字符串，大小写字母视为不同)*

引用单元：StrUtils

函数声明：function AnsiReplaceStr(const AText, AFromText, AToText: string): string;



```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  AText, AFromText, AToText: string;
  Tmp : String;
begin
  //uses StrUtils
  AText := 'I love the C++ , C++ and C++.';
  AFromText := 'C++';
  AToText:= 'Delphi';
  //字母大小写被视为不同
  Tmp := AnsiReplaceStr(AText,AFromText,AToText);
  Showmessage('替换过的字符串为:' + tmp);
End;
```

AnsiReplaceText(替换指定的子字符串，大小写字母视为相同)

引用单元：StrUtils
函数声明：function AnsiReplaceText(const AText, AFromText, AToText: string): string;

例 E-22

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AText, AFromText, AToText: string;
  Tmp : String;
begin
  //uses StrUtils
  AText := 'I love the C++ C++ C++.';
  AFromText := 'c++';
  AToText:= 'Delphi';
  //字母大小写被视为相同
  Tmp := AnsiReplaceText(AText,AFromText,AToText);
  Showmessage('替换过的字符串为:' + tmp);
End;
```

MidStr、LeftStr 与 RightStr(截取子字符串)

引用单元：StrUtils
函数声明：function MidStr(const AText: string; const AStart, ACount: Integer): string; function LeftStr(const AText: string; ACount: Integer): string; function RightStr(const AText: string; ACount: Integer): string;

例 E-23

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   Str : string;
begin
   Str := 'ABCDEFGHIJKLMNOP';
   //CDEF
   Showmessage('MidStr(Str,3,4) = ' + MidStr(Str,3,4));
   //ABCD
   Showmessage('LeftStr(Str,4)  = ' + LeftStr(Str,4));
   //MNOP
   Showmessage('RightStr(Str,4) = ' + RightStr(Str,4));
End;
```

Pos(查找子字符串在字符串中的位置)

引用单元：System

函数声明：function Pos(Substr: string; S: string): Integer;

范例 E-24

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S: string;
   I:Integer;
begin
  //字符串 S 中有 4 个全角空格
  S := '  Delphi   is   good  ';
  //用下划线替换字符串中的全角空格
  while Pos('  ', S) > 0 do
  begin
    I:=Pos('  ', S);
    S[I]:= '_';
    S[I+1]:='_';
  end;
  ShowMessage(S);
End;
```

AnsiPos(查找子字符串在字符串中的位置)

引用单元：SysUtils

函数声明：function AnsiPos(const Substr, S: string): Integer;

范例 E-25

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  S: strirg;
  I:Integer;
begin
  //字符串 S 中有 4 个全角空格
  S := '  Delphi   is   good  ';
  i := AnsiPos('Delphi',S);
  //全角被视为两个字符，所以是第 3 个字
  ShowMessage('子字符串出现在第' + IntToStr(i) + '字');
End;
```

Length(返回指定字符串的长度)

引用单元：System

函数声明：function Length(S): Integer;

E-26

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  i:Integer;
begin
  //字符串 S 中有 4 个全角空格及两个半角空格
  S := '  Delphi   is   good  ';
  i := Length(S);
  //全角被视为两个字符，所以是 22 个字
  ShowMessage('字符串长度为:' + IntToStr(i));
End;
```

Copy(字符串复制)

引用单元：System

函数声明：function Copy(S; Index, Count: Integer): string;

E-27

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  S := 'Delphi is good!!';
  S := Copy(S, 1, 8);
  ShowMessage(S); //Delphi i
end;
```

Delete(字符串删除)

引用单元：System

函数声明：procedure Delete(var S: string; Index, Count:Integer);

范例 E-28

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  S := 'Delphi is good!!';
  Delete(S, 1, 8);
  ShowMessage(S); //s good!!
end;
```

Concat(字符串相加)

引用单元：System

函数声明：function Concat(s1 [, s2,..., sn]: string): string;

范例 E-29

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S1,S2,S3,Str:String;
begin
   S1:='今天';
   S2:='下雨';
   S3:='吗?';
   Str :=Concat(S1,S2,S3); //等于Str:=S1+S2+S3;
   ShowMessage(Str);
end;
```

Insert(将某个字符串插入另一个字符串指定位置)

引用单元：System

函数声明：procedure Insert(Source: string; var S: string; Index: Integer);

范例 E-30

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  S := 'I love !!';
  Insert('Delphi ', S, 8);
```

```
  showmessage(S); //I love Delphi !!
end;
```

- *AnsiStartsStr(判断子字符串是否为母字符串的开头)*

引用单元：StrUtils

函数声明：function AnsiStartsStr(const ASubText, AText: string): Boolean;



E-31

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, SubString : string;
begin
  //uses StrUtils
  S := 'Delphi is good!!';
  SubString := 'Delphi';
  //字母大小写视为不同
  if AnsiStartsStr(SubString, S) then
     showmessage('子字符串为母字符串的开头');
end;
```

- *AnsiStartsText(判断子字符串是否为母字符串的开头)*

引用单元：StrUtils

函数声明：function AnsiStartsText(const ASubText, AText: string): Boolean;



E-32

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, SubString : string;
begin
  //uses StrUtils
  S := 'Delphi is good!!';
  SubString := 'delphi';
  //字母大小写视为相同
  if AnsiStartsText(SubString, S) then
     showmessage('子字符串为母字符串的开头');
end;
```

- *AnsiEndsStr(判断子字符串是否为母字符串的结尾)*

引用单元：StrUtils

函数声明：function AnsiEndsStr(const ASubText, AText: string): Boolean;

范例 E-33

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, SubString : string;
begin
  //uses StrUtils
  S := 'I love Delphi';
  SubString := 'Delphi';
  //字母大小写视为不同
  if AnsiEndsStr(SubString, S) then
     showmessage('子字符串为母字符串的结尾');
end;
```

- *AnsiEndsText(判断子字符串是否为母字符串的结尾)*

引用单元：StrUtils

函数声明：function AnsiEndsText(const ASubText, AText: string): Boolean;

范例 E-34

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, SubString : string;
begin
  //uses StrUtils
  S := 'I love Delphi';
  SubString := 'delphi';
  //字母大小写视为相同
  if AnsiEndsText(SubString, S) then
     showmessage('子字符串为母字符串的结尾');
end;
```

- *AnsiIndexStr(返回指定字符串为字符串数组中的位置)*

引用单元：StrUtils

函数声明：function AnsiIndexStr(const AText: string; const AValues: array of string): Integer;

范例 E-35

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, S1,S2,S3 : string;
  i : integer;
```

```
begin
  //uses StrUtils
  S :- 'Delphi';
  S1 := 'TEST';
  S2 := 'Delphi';
  S3 := 'Nothing';
  //字母大小写视为不同
  i := AnsiIndexStr(S,[S1,S2,S3]);
  //从 0 开始算起
  showmessage('指定字符串在数组中的第'+ IntToStr(i) +'个子字符串');
end;
```

AnsiIndexText(返回指定字符串为字符串数组中的位置)

引用单元：StrUtils

函数声明：function AnsiIndexText(const AText: string; const AValues: array of string): Integer;

范例 E-36

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, S1,S2,S3 : string;
  i : Integer;
begin
  //uses StrUtils
  S := 'Delphi';
  S1 := 'TEST';
  S2 := 'delphi';
  S3 := 'Nothing';
  //字母大小写视为相同
  i := AnsiIndexText(S,[S1,S2,S3]);
  //从 0 开始算起
  showmessage('指定字符串在数组中的第'+ IntToStr(i) +'个子字符串');
end;
```

QuotedStr(可为指定字符串加上一组单引号)

引用单元：StrUtils

函数声明：function QuotedStr(const S: string): string;

范例 E-37

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S : string;
```

```
begin
  //uses StrUtils
  S := 'Delphi';
  S := QuotedStr(S);
  showmessage(S); //'Delphi'
end;
```

AnsiQuotedStr(可为指定字符串加上一组指定的字符于字符串的前后)

引用单元：StrUtils

函数声明：function AnsiQuotedStr(const S: string; Quote: Char): string;

范例 E-38

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S : string;
begin
  //uses StrUtils
  S := 'Delphi';
  S := AnsiQuotedStr(S,'@');
  showmessage(S); //@Delphi@
end;
```

AnsiDequotedStr(可为指定字符串的前后去除一个指定的字符)

引用单元：StrUtils

函数声明：function AnsiQuotedStr(const S: string; Quote: Char): string;

范例 E-39

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S : string;
begin
  //uses StrUtils
  S := '@Delphi@';
  S := AnsiDequotedStr(S,'@');
  showmessage(S); //Delphi
end;
```

IsDelimiter(判断指定的子字符串是否出现在指定的位置)

引用单元：StrUtils

函数声明：function IsDelimiter(const Delimiters, S: string; Index: Integer): Boolean;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  De_limiters, S : string;
begin
  Delimiters := 'pxxx'; //这样符号条件
  //Delimiters := 'Phi'; //这样不符号条件
  S := 'Delphi';
  //只比较 Delimiters 的第一个字母而且大小写视为不同
  //而且从 0 开始计算
  if  IsDelimiter(Delimiters, S, 4) then
      showmessage('指定的子字符串出现在您所指定的位置');
end;
```

LastDelimiter(指定的子字符串中任何一个字符出现在指定字符串中的最后一个位置)

引用单元：SysUtils

函数声明：function LastDelimiter(const Delimiters, S: string): Integer;



```
procedure TForm1.Button1Click(Sender: TObject);
var
  Delimiters, S : string;
  i : Integer;
begin
  Delimiters := 'xpx';
  S := 'Delphi';
  //只比较 Delimiters 的第一个字母而且大小写视为不同
  //而且从 1 开始计算
  i := LastDelimiter(Delimiters, S);
  Showmessage(Inttostr(i)); //p 在第 4 个字出现
end;
```

Str(将数值转换为指定格式的字符串)

引用单元：System

函数声明：procedure Str(X [: Width [: Decimals]]; var S);



```
procedure TForm1.Button1Click(Sender: TObject);
const
  W=12; //代表数值的位数(包含小数点)，如果位数小于所要转换的数值则忽略
```

```
  D=2;  //代表小数字数
var
  S: string;
begin
  STr(1234678.5678:W:D,S);
  ShowMessage('''+S+'''); {' 1234678.56'}
end;
```

- *Val(将字符串转为数字)*

引用单元：System

函数声明：procedure Val(S; var V; var Code: Integer);

范例 E-43

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, Code: Integer;
begin
  //将 Edit1.Text 的字符串，存入 I，Code 表示转换过程中发生位
  Val(Edit1.Text, I, Code);
  if code <> 0 then  //发生错误
    ShowMessage('转换错误发生在第'+ IntToStr(Code)+'位')
  else
    ShowMessage('转换结果为:'+ IntToStr(I));
end;
```

- *StringOfChar(返回指定个数的字符)*

引用单元：System

函数声明：function StringOfChar(Ch: Char; Count: Integer): string;

范例 E-44

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S : String;
  C : Char;
begin
  C := 'D';
  S := StringOfChar(C,10);
  ShowMessage(S);  //DDDDDDDDDD
end;
```

StuffString(在指定位置与长度插入指定的子字符串)

引用单元：System

函数声明：function StuffString(const AText: string; AStart, ALength: Cardinal; const ASubText: string): string;

范例 E-45

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AText,ASubText, Str : String;
  AStart, ALength: Cardinal;
begin
  //uses StrUtils
  AText := 'ABCDEFGHIJKLMNP';
  ASubText := '1234567890';
  AStart := 2;  //插入的位置
  ALength := 5; //清除的个数
  Str := StuffString(AText,AStart,ALength,ASubText);
  ShowMessage(Str);  //A1234567890GHIJKLMNP
end;
```

WrapText(将指定字符串依照指定长度加以分隔成多行的字符串)

引用单元：System

函数声明：function WrapText(const Line, BreakStr: string; nBreakChars: TSysCharSet; MaxCol: Integer):string;

范例 E-46

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str : String;
begin
  Str := '0123456789 0123456789 0123456789 0123456789';
  //其中 "#10" 为分行的字符
  //['9',' ']为可分割的字符
  //11 代表每一行最长的限制
  Str := WrapText(Str,#10,['9',' '], 11);
  Showmessage(Str);
{ //执行结果如下:
0123456789
0123456789
0123456789
```

```
0123456789
}
end;
```

- *SetLength (动态更改动态字符串的长度)*

引用单元：System
函数声明：procedure SetLength(var S; NewLength: Integer);

- *DupeString (重复指定的字符串)*

引用单元：System
函数声明：function DupeString(const AText: string; ACount: Integer): string;

例 E-47

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str : String;
begin
  //uses StrUtils
  Str := 'Delphi';
  Str := DupeString(Str,3);
  Showmessage(Str); //DelphiDelphiDelphi
end;
```

- *ReverseString (指定的字符串加以反转)*

引用单元：StrUtils
函数声明：function ReverseString(const AText: string): string;

例 E-48

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str : String;
begin
  //uses StrUtils
  Str := '0123456789';
  Str := ReverseString(Str);
  Showmessage(Str); //9876543210
end;
```

E.2 以 Null 结尾的字符串处理函数

本小节将为您介绍 Delphi 6 所提供的以 Null 字符为结尾的字符串处理函数。那什么是以 Null 字符为结尾的字符串呢？所谓以 Null 字符为结尾的字符串代表的是一个以 Null(#0)结尾，而且其索引值从 0 开始计算的字符串。因为这种字符串并没有固定长度，因此就以第一个 Null 字符来做为字符串的结束。在处理以 Null 字符为结尾的字符串过程中，经常会配合字符串常数 PChar 与 PWideChar 一起使用。其分别代表指向以 Null 字符结尾的 Char 和 WideChar 数组指针。

本小节将为您介绍 Delphi 6 所提供的以 Null 字符为结尾的字符串处理函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 E.2 所示。

表 E.2

函数名称	单元文件	所代表的意义
StrUpper	SysUtils	此函数可以指定字符串中的所有字符转为大写字母
AnsiStrUpper	SysUtils	此函数可以指定字符串中的所有字符转为大写字母，但其支持国际字符(international characters)
StrLower	SysUtils	此函数可以指定字符串中的所有字符转为小写字母
AnsiStrLower	SysUtils	此函数可以指定字符串中的所有字符转为小写字母，但其支持国际字符(international characters)
StrCopy	SysUtils	此函数可进行两个字符串的复制，复制时并不会检查内存的大小。目的字符串的长度至少是来源字符串的长度加 1
StrECopy	SysUtils	此函数可进行字符串的复制并返回字符串结束地址。但复制时并不会检查内存的大小，目的字符串的长度至少是来源字符串的长度加 1
StrLCopy	SysUtils	此函数可进行指定长度的字符串复制，目的字符串的长度至少是来源字符串的长度加 1
StrPCopy	SysUtils	此函数可将 String 类型的字符串复制到 PChar 类型字符串变量
StrPLCopy	SysUtils	此函数可将指定长度的 String 类型字符串复制到 PChar 类型字符串变量
StrPos	SysUtils	此函数可返回子字符串在指定字符串中的位置，如果找不到子字符串则返回 nil 值，否则返回找到的位置
AnsiStrPos	SysUtils	此函数可返回子字符串在指定字符串中的位置。如果找不到子字符串则返回 nil 值，否则返回找到的位置。此函数支持国际字符(international characters)
StrScan	SysUtils	此函数可返回指定字符在某字符串中出现的位置
StrRScan	SysUtils	此函数可返回指定字符在某字符串中最后出现的位置
AnsiStrScan	SysUtils	此函数可返回指定字符在某字符串中出现的位置。此函数支持国际字符(international characters)

续上表

函数名称	单元文件	所代表的意义
AnsiStrRScan	SysUtils	此函数可返回指定字符在某字符串中最后出现的位置。此函数支持国际字符(international characters)
StrComp	SysUtils	此函数可比较两字符串的大小，比较过程中将字母大小写视为不同
StrIComp	SysUtils	此函数可比较两字符串的大小，比较过程中将字母大小写视为相同
StrLComp	SysUtils	此函数可比较指定长度的字符串，比较过程中大小写字母视为不同
StrLIComp	SysUtils	此函数可比较指定长度的字符串，比较过程中大小写字母视为相同
AnsiStrComp	SysUtils	此函数可比较两字符串的大小，比较过程中将字母大小写视为不同。此函数支持国际字符(international characters)
AnsiStrIComp	SysUtils	此函数可比较两字符串的大小，比较过程中将字母大小写视为相同。此函数支持国际字符(international characters)
AnsiStrLComp	SysUtils	此函数可比较指定长度的字符串，比较过程中大小写字母视为不同。此函数支持国际字符(international characters)
AnsiStrLIComp	SysUtils	此函数可比较指定长度的字符串，比较过程中大小写字母视为相同。此函数支持国际字符(international characters)
StrEnd	SysUtils	此函数可获取指定字符串的最后一个字符位置
StrLen	SysUtils	此函数可返回指定字符串的长度
StrMove	SysUtils	此函数可复制指定字符数量的字符串
StrCat	SysUtils	此函数可将指定的子字符串相加为新的字符串
StrLCat	SysUtils	此函数可将指定长度的子字符串相加为新的字符串
StrPas	SysUtils	此函数可以 PChar 变量转为 String 类型
NewStr	SysUtils	此函数可以为字符串指针分配新的内存
DisposeStr	SysUtils	此函数可释放字符串指针的内存空间
StrNew	SysUtils	此函数可为字符串指针变量分配内存空间
StrDispose	SysUtils	此函数可释放由 StrAlloc 函数或 StrNew 函数所分配的内存
StrAlloc	SysUtils	此函数可为字符串指针变量分配内存空间
StrBufSize	SysUtils	此函数可返回由 StrAlloc 函数所分配的内存大小

以下将以范例的方式说明每一个函数的使用方法。

StrUpper(将所有字符转为大写字母)

引用单元：SysUtils
函数声明：function StrUpper(Str: PChar): PChar;

范例 E-49

```
procedure TForm1.Button1Click(Sender: TObject);
const
    //先声明一个字符数组常数
    TempString : array[0..20] of Char = 'Delphi is good!!';
var
    //声明一个字符串指针变量
    MyString : PChar;
begin
    //将字符串指针变量赋值字符数组
    MyString := @TempString;
    //转换为大写字母
    MyString := AnsiStrUpper(MyString);
    ShowMessage(MyString); //以 Pchar 类型显示
    ShowMessage(String(MyString)); //以 String 类型显示
    //结果为 DELPHI IS GOOD!!
end;
```

AnsiStrUpper(将所有字符转为大写字母)

引用单元：SysUtils
函数声明：function AnsiStrUpper(Str: PChar): PChar;

范例 E-50

```
procedure TForm1.Button1Click(Sender: TObject);
const
    TempString : array[0..20] of Char = 'Delphi is good!!';
var
    MyString : PChar;
begin
    MyString := @TempString;
    //转换为大写字母
    MyString := AnsiStrUpper(MyString);
    ShowMessage(MyString); //以 Pchar 类型显示
    //结果为 DELPHI IS GOOD!!
end;
```

StrLower(将所有字符转为小写字母)

引用单元：SysUtils
函数声明：function StrLower(Str: PChar): PChar;

例 E-51

```
procedure TForm1.Button1Click(Sender: TObject);
const
   TempString : array[0..20] of Char = 'Delphi is good!!';
var
   MyString : PChar;
begin
   MyString := @TempString;
   //转换为小写字母
   MyString := StrLower(MyString);
   ShowMessage(MyString); //以 Pchar 类型显示
   //结果为 delphi is good!!
end;
```

- *AnsiStrLower(将所有字符转为小写字母)*

引用单元：SysUtils

函数声明：function AnsiStrLower(Str: PChar): PChar;

例 E-52

```
procedure TForm1.Button1Click(Sender: TObject);
const
   TempString : array[0..20] of Char = 'Delphi is good!!';
var
   MyString : PChar;
begin
   MyString := @TempString;
   //转换为小写字母
   MyString := AnsiStrLower(MyString);
   ShowMessage(MyString); //以 Pchar 类型显示
   //结果为 delphi is good!!
end;
```

- *StrCopy(进行两个字符串的复制)*

引用单元：SysUtils

函数声明：function StrCopy(Dest, Source: PChar): PChar;

例 E-53

```
procedure TForm1.Button1Click(Sender: TObject);
var
  //当 S 的长度小于所要复制的字符串会发生错误
```

```
  S: array[0..20] of Char;
  Temp : PChar;
begin
  Temp := 'Delphi is good!!';
  StrCopy(S,Temp);
  ShowMessage(S);
end;
```

StrECopy(字符串复制并返回字符串结束地址)

引用单元：SysUtils

函数声明：function StrECopy(Dest, Source: PChar): PChar;



E-54

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi ';
  S2: PChar = 'is good!!';
var
  S: array[0..20] of Char;
begin
  //先复制 S1 到 S，并返回字符串结束地址，从结束地址再接上 S2
  StrECopy(StrECopy(S,S1),S2);
  showmessage(S);
end;
```

StrLCopy(进行指定长度的字符串复制)

引用单元：SysUtils

函数声明：function StrLCopy(Dest, Source: PChar; MaxLen: Cardinal): PChar;



E-55

```
procedure TForm1.Button1Click(Sender: TObject);
const
  Temp : Pchar = 'Delphi is good and I love it so much!!';
var
  //当 S 的长度小于所要复制的字符串会发生错误
  S: array[0..20] of Char;
begin
  //限制复制的长度，以免发生长度不够的错误
  StrLCopy(S, Temp ,SizeOf(S)-1); //只能复制原长度-1 的字符串
  ShowMessage(S);
end;
```

StrPCopy(复制 String 类型字符串到 PChar 类型字符串变量)

引用单元：SysUtils

函数声明：function StrPCopy(Dest: PChar; const Source: string): PChar;

范例 E-56

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array[0..20] of Char;
  S1: String;
begin
  S1 := 'Delphi is good!!';
  StrPCopy(S,S1);
  showmessage(S);
end;
```

StrPLCopy(复制指定长度的 String 类型字符串到 Pchar 类型字符串变量)

引用单元：SysUtils

函数声明：function StrPLCopy(Dest: PChar; const Source: string; MaxLen: Cardinal): PChar;

范例 E-57

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array[0..20] of Char;
  S1: String;
begin
  S1 := 'Delphi is good!!';
  StrPLCopy(S,S1,10);
  showmessage(S);
end;
```

StrPos(返回子字符串在指定字符串中的位置)

引用单元：SysUtils

函数声明：function StrPos(Str1, Str2: PChar): PChar;

函数结果：如果找不到就返回 nil 值，否则返回找到的位置

范例 E-58

```
procedure TForm1.Button1Click(Sender: TObject);
const
```

```
  S1: PChar = 'Delphi is good!!';
  S2: PChar = 'is';
var
  ReturnPoint:PChar;
begin
  ReturnPoint:= StrPos(S1,S2);
  if ReturnPoint = nil then
     ShowMessage('找不到指定的字符串:'+S2)
  else
 ShowMessage(S2+'字符串出现在第'+IntToStr(ReturnPoint-S1)+'个字');
end;
```

AnsiStrPos(返回子字符串在指定字符串中的位置)

引用单元：SysUtils

函数声明：function AnsiStrPos(Str, SubStr: PChar): PChar;

函数结果：如果找不到就返回 nil 值，否则返回找到的位置

E-59

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi is good!!';
  S2: PChar = 'is';
var
  ReturnPoint:PChar;
begin
  ReturnPoint:= AnsiStrPos(S1,S2);
  if ReturnPoint = nil then
     ShowMessage('找不到指定的字符串:'+S2)
  else
 ShowMessage(S2+'字符串出现在第'+IntToStr(ReturnPoint-S1)+'个字');
end;
```

StrScan(返回指定字符在某字符串中出现的位置)

引用单元：SysUtils

函数声明：function StrScan(Str: PChar; Chr: Char): PChar;

E-60

```
Procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi is good!!';
  S2: Char = 'i';
```

```
var
  ReturnPoint:PChar;
begin
  //只适用于Char
  ReturnPoint:= StrScan(S1,S2);
  if ReturnPoint <> nil then
    ShowMessage('找不到指定的字符串:'+S2)
  else
 ShowMessage(S2+'字符串出现在第'+IntToStr(ReturnPoint-S1)+'个字');
end;
```

- *StrRScan(返回指定字符在某字符串出现的最后一个位置)*

引用单元：SysUtils

函数声明：function StrRScan(Str: PChar; Chr: Char): PChar;

例 E-61

```
Procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi is good!!';
  S2: Char = 'i';
Var
  ReturnPoint:PChar;
Begin
  //只适用于Char
  ReturnPoint:= StrRScan(S1,S2);
  ShowMessage('最后一个'+S2+'出现在第'+IntToStr(
                                    ReturnPoint-S1)+'个字');
end;
```

- *AnsiStrScan(返回指定字符在某字符串中出现的位置)*

引用单元：SysUtils

函数声明：function AnsiStrScan(Str: PChar; Chr: Char): PChar;

例 E-62

```
Procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi is good!!';
  S2: Char = 'i';
var
  ReturnPoint:PChar;
begin
```

```
  //只适用于 Char
  ReturnPoint:= AnsiStrScan(S1,S2);
  if ReturnPoint = nil then
    ShowMessage('找不到指定的字符串:'+S2)
  else
 ShowMessage(S2+'字符串出现在第'+IntToStr(ReturnPoint-S1)+'个字');
end;
```

- *AnsiStrRScan(返回指定字符在某字符串出现的最后一个位置)*

引用单元：SysUtils
函数声明：function AnsiStrRScan(Str: PChar; Chr: Char): PChar;

E-63

```
Procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi is good!!';
  S2: Char = 'i';
Var
  ReturnPoint:PChar;
Begin
  //只适用于 Char
  ReturnPoint:= AnsiStrRScan(S1,S2);
ShowMessage('最后一个'+S2+'出现在第'+IntToStr(ReturnPoint-S1)+'个字');
end;
```

- *StrComp(比较两字符串的大小，字母大小写视为不同)*

引用单元：SysUtils
函数声明：function StrComp(Str1, Str2 : PChar): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

E-64

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi';
  S2: PChar = 'delphi';
var
  ResultFlag: Integer;
begin
  //将字母大小写视为不同
```

```
  ResultFlag:=StrComp(S1, S2);
  if ResultFlag < 0 then ShowMessage('Delphi < delphi')
  else if ResultFlag > 0 then ShowMessage('Delphi > delphi')
  else ShowMessage('Delphi = delphi ');
end;
```

- *StrIComp(比较两字符串的大小，字母大小写视为相同)*

引用单元：SysUtils
函数声明：function StrIComp(Str1, Str2:PChar): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

例 E-65

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi';
  S2: PChar = 'delphi';
var
  ResultFlag: Integer;
begin
  //将字母大小写视为相同
  ResultFlag:=StrIComp(S1, S2);
  if ResultFlag < 0 then ShowMessage('Delphi < delphi')
  else if ResultFlag > 0 then ShowMessage('Delphi > delphi')
  else ShowMessage('Delphi = delphi ');
end;
```

- *StrLComp(比较指定长度的字符串,大小写字母视为不同)*

引用单元：SysUtils
函数声明：function StrLComp(Str1, Str2: PChar; MaxLen: Cardinal): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

例 E-66

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'abcDEF';
  S2: PChar = 'abcdef';
```

```
var
  ResultFlag,I: Integer;
begin
  I:=3;
  ResultFlag:=StrLComp(S1, S2,I);
  if ResultFlag < 0 then ShowMessage(S1+'<'+S2)
  else if ResultFlag > 0 then ShowMessage(S1+'>'+S2)
  else ShowMessage(S1+'='+S2);
end;
```

- *StrLIComp(比较指定长度的字符串,大小写字母视为相同)*

引用单元：SysUtils
函数声明：function StrLIComp(Str1, Str2: PChar; MaxLen: Cardinals): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

范例 E-67

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'abcDEF';
  S2: PChar = 'abcdef';
var
  ResultFlag,I: Integer;
begin
  I:=5;
  ResultFlag:=StrLIComp(S1, S2,I);
  if ResultFlag < 0 then ShowMessage(S1+'<'+S2)
  else if ResultFlag > 0 then ShowMessage(S1+'>'+S2)
  else ShowMessage(S1+'='+S2);
end;
```

- *AnsiStrComp(比较两字符串的大小，字母大小写视为不同)*

引用单元：SysUtils
函数声明：function AnsiStrComp(S1, S2: PChar): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

范例 E-68

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
const
  S1: PChar = 'Delphi';
  S2: PChar = 'delphi';
var
  ResultFlag: Integer;
begin
  //将字母大小写视为不同
  ResultFlag:=AnsiStrComp(S1, S2);
  if ResultFlag < 0 then ShowMessage('Delphi < delphi')
  else if ResultFlag > 0 then ShowMessage('Delphi > delphi')
  else ShowMessage('Delphi = delphi ');
end;
```

- *AnsiStrIComp(比较两字符串的大小，字母大小写视为相同)*

引用单元：SysUtils
函数声明：function AnsiStrIComp(S1, S2: PChar): Integer;
返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0



E-69

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi';
  S2: PChar = 'delphi';
var
  ResultFlag: Integer;
begin
  //将字母大小写视为相同
  ResultFlag:= AnsiStrIComp(S1, S2);
  if ResultFlag < 0 then ShowMessage('Delphi < delphi')
  else if ResultFlag > 0 then ShowMessage('Delphi > delphi')
  else ShowMessage('Delphi = delphi ');
end;
```

- *AnsiStrLComp(比较指定长度的字符串，大小写字母视为不同)*

引用单元：SysUtils
函数声明：function AnsiStrLComp(S1, S2: PChar; MaxLen: Cardinal): Integer;
返回结果：返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

范例 E-70

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'abcDEF';
  S2: PChar = 'abcdef';
var
  ResultFlag,I: Integer;
begin
  I:=3;
  ResultFlag:= AnsiStrLComp(S1, S2,I);
  if ResultFlag < 0 then ShowMessage(S1+'<'+S2)
  else if ResultFlag > 0 then ShowMessage(S1+'>'+S2)
  else ShowMessage(S1+'='+S2);
end;
```

- *AnsiStrLIComp*(比较指定长度的字符串,大小写字母视为相同)

引用单元：SysUtils
函数声明：function AnsiStrLIComp(S1, S2: PChar; MaxLen: Cardinal): Integer;
返回结果：返回结果：当 S1 > S2 返回大于 0 的整数 当 S1 < S2 返回小于 0 的整数 当 S1 = S2 返回 0

范例 E-71

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'abcDEF';
  S2: PChar = 'abcdef';
var
  ResultFlag,I: Integer;
begin
  I:=5;
  ResultFlag:= AnsiStrLIComp(S1, S2,I);
  if ResultFlag < 0 then ShowMessage(S1+'<'+S2)
  else if ResultFlag > 0 then ShowMessage(S1+'>'+S2)
  else ShowMessage(S1+'='+S2);
end;
```

- *StrEnd*(获取指定字符串的最后一个字符位置)

引用单元：SysUtils
函数声明：function StrEnd(const Str: PChar): PChar;

E-72

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str : String;
  Temp : PChar;
  Position: PChar;
begin
  //此范例可以 Str 变量的内容，用字符串指针的方式加以反转
  Str := '0123456789';
  //获取一块可存放 Str 变量内容的内存空间
  GetMem(Temp, Length(Temp)+1);
  //进行复制
  StrCopy(Temp, PChar(Str));
  //获取字符串的最后一个内存位置
  Position := StrEnd(Temp);
  Str:= '';
  //判断当前指针是否为 Temp 的开头
  while Position >= Temp do
  begin
    //指针往后退一个字
    Position := Position - 1;
    //将当前所指的字累加到 Str 变量
    Str := Str + Position^;
  end;
  //释放内存
  FreeMem(Temp);
  //显示结果
  Showmessage(Str);
end;
```

StrLen(字符串长度)

引用单元：SysUtils

函数声明：function StrLen(Str: PChar): Cardinal;

E-73

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Str:PChar;
begin
  Str:='Delphi 赞 '; //中文算两个 Bytes，长度包含空格
  showmessage(IntToStr(StrLen(Str)));
end;
```

StrMove(复制指定字符数量的字符串)

引用单元：SysUtils
函数声明：function StrMove(Dest, Source: PChar; Count: Cardinal): PChar;

范例 E-74

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S1:PChar;
  S: array[0..20] of Char;
begin
  S1:='Delphi is good!!';
  StrMove(S,S1,6);
  ShowMessage('S='+S+',S1='+S1); //S=Delphi,S1 不变
end;
```

StrCat(将指定的子字符串相加)

引用单元：SysUtils
函数声明：function StrCat(Dest, Source: PChar): PChar; 参数说明:Dest 是所要存放的变量、Source 是被复制的字符串，Dest 的内存至少要为 StrLen(Dest)+StrLen(Source)+1，否则程序会出错，函数返回值表示前的字符串相加情况。

范例 E-75

```
procedure TForm1.Button1Click(Sender: TObject);
const
   S1: PChar = 'Delphi ';  //长度为 7
   S2: PChar = 'is good!!';//长度为 9
var
   S: array[0..17] of Char;  //声明一个长度为 17 的字符串
begin
  Canvas.TextOut(10,10,StrCat(S,S1)); //显示 Delphi
  Canvas.TextOut(10,30,StrCat(S,S2)); //显示 Delphi is good!!
  //以下会发生错误，因为超过长度
  //Canvas.TextOut(10,30,StrCat(S,'Abc'));
end;
```

StrLCat(将指定长度字符串相加)

引用单元：SysUtils
函数声明：function StrLCat(Dest, Source: PChar; MaxLen: Cardinal): PChar;

范例 E-76

```
procedure TForm1.Button1Click(Sender: TObject);
const
  S1: PChar = 'Delphi '; //长度为 7
  S2: PChar = 'is good!!';//长度为 9
 var
  S: array[0..17] of Char;  //声明一个长度为 18 的字符串
begin
  Canvas.TextOut(10,10,StrLCat(S,S1,sizeof(S))); //Delphi
  Canvas.TextOut(10,30,StrLCat(S,S2,sizeof(S))); //Delphi is good!!
  //以下会发生错误，因为超过长度
  //Canvas.TextOut(10,50,StrLCat(S,'Abc',sizeof(S)));
end;
```

- *StrPas(将 Pchar 转为 String)*

引用单元：SysUtils
函数声明：function StrPas(Str: PChar): string;

```
procedure TForm1.Button1Click(Sender: TObject);
var
    S1:PChar;
    S2:String;
begin
   S1:= 'Delphi    ';
   S2:=StrPas(S1);
   ShowMessage(S2);
end;
```

- *NewStr(为字符串指针分配新内存)*

引用单元：SysUtils
函数声明：function NewStr(const S: string): PString;

- *DisposeStr(释放字符串指针的内存)*

引用单元：SysUtils
函数声明：procedure DisposeStr(P: PString);

```
procedure TForm1.Button1Click(Sender: TObject);
var
   S:String;      //声明一个 String 变量
   PS:PString;    //声明一个 PString 变量
Begin
   S:='Delphi     is good!!';
```

```
  PS:=NewStr(S);   //创建一个新的字符串
  ShowMessage(String(S));
  DisposeStr(PS);  //将这个字符串释放
end;
```

- *StrNew(分配字符串内存)*

引用单元：SysUtils
函数声明：function StrNew(Str: PChar): PChar;

- *StrDispose(释放由 StrAlloc 或 StrNew 所分配的内存)*

引用单元：SysUtils
函数声明：procedure StrDispose(Str: PChar);

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S:PChar;     //声明一个 PChar 变量
  PS:PChar;    //声明一个 PChar 变量
begin
  S:='Delphi is good!!';
  PS:=StrNew(S);   //创建一个新的字符串
  ShowMessage(String(S));
  StrDispose(PS);  //将这个字符串释放
end;
```

- *StrAlloc(分配字符串内存)*

引用单元：SysUtils
函数声明：function StrAlloc(Size: Cardinal): PChar;

- *StrBufSize(返回由 StrAlloc 分配的内存大小)*

引用单元：SysUtils
函数声明：function StrBufSize(Str: PChar): Cardinal;

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Size:Cardinal;
   Str:PChar;
begin
   Size:=20;
   Str:=StrAlloc(Size);
   ShowMessage(IntToStr(StrBufSize(Str)));
end;
```

E.3 数值处理函数

本小节将为您介绍 Delphi 6 所提供的数值处理函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 E.3 所示。

表 E.3

函数名称	单元文件	所代表的意义
Abs	System	此函数可返回指定整数或实数的绝对值
ArcTan	System	此函数可返回一角函数 ArcTan 值
Cos	System	此函数可返回三角函数 Cos 值
Sin	System	此函数可返回三角函数 Sin 值
Frac	System	此函数可返回实数的小数部分
Int	System	此函数可返回实数的整数部分
Round	System	此函数可将实数四舍五入转为整数
Trunc	System	此函数可将实数小数无条件舍去转为整数
Pi	System	此函数可返回 π 值
Sqr	System	此函数可返回指定实数的平方
Sqrt	System	此函数可返回指定实数的平方根
Ln	System	此函数可返回自然对数值
Exp	System	此函数可返回指定实数的指数
FloatToDecimal	SysUtils	此函数可将指定的浮点数转为十进制数
FloatToText	SysUtils	此函数可将浮点数转换为指定格式的十进制数
FloatToTextFmt	SysUtils	此函数可将浮点数转换为指定格式的十进制数
FloatToStr	SysUtils	此函数可将浮点数转为字符串
FloatToStrF	SysUtils	此函数可将浮点数转为指定格式的字符串
FormatFloat	SysUtils	此函数可将浮点数转为指定格式的字符串
Random	System	此函数可返回一个随机数
Randomize	System	此函数可启用随机数种子
Dec	System	此函数可将指定的变量值递减
Inc	System	此函数可将指定的变量值递增
Odd	System	此函数可检查指定的变量是否为奇数
IntToHex	SysUtils	此函数可将指定的十进制数值转为十六进制
IntToStr	SysUtils	此函数可将十进制整数转为字符串
Chr	System	此函数可返回指定字符的 ASCII 码

以下将以范例的方式说明每一个函数的使用方法。

- *Abs(绝对值)*

引用单元：System
函数声明：function Abs(X:整数或实数);



E-77

```
procedure TForm1.Button1Click(Sender: TObject);
var
  V1: Real;
  V2: Integer;
begin
  V1 := Abs(-9.9); {9.9}
  V2 := Abs(-9);   {9}
end;
```

- *ArcTan(三角函数)*

引用单元：System
函数声明：function ArcTan(X: Extended): Extended;

- *Cos(三角函数)*

引用单元：System
函数声明：function Cos(X: Extended): Extended;

- *Sin(三角函数)*

引用单元：System
函数声明：function Sin(X: Extended): Extended;



E-78

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowMessage(FloatToStr(ArcTan(1)));
   ShowMessage(FloatToStr(sin(1)));
   ShowMessage(FloatToStr(cos(1)));
end;
```

- *Frac(返回实数的小数部分)*

引用单元：System
函数声明：function Frac(X: Extended): Extended;

- *Int(返回实数的整数部分)*

引用单元：System
函数声明：function Int(X: Extended): Extended;

- *Round(将实数四舍五入转为整数)*

引用单元：System
函数声明：function Round(X: Extended): Int64;

- *Trunc(将实数小数无条件舍去转为整数)*

引用单元：System
函数声明：function Trunc(X: Extended): Int64;



E-79

```
procedure TForm1.Button1Click(Sender: TObject);
var Str:String;
begin
  Str:='整数:'+FloatToStr(Int(123.56))+#10;
  Str:=Str+'小数:'+FloatToStr(Frac(123.56))+#10;
  Str:=Str+'四舍五入为:'+FloatToStr(Round(123.56))+#10;
  Str:=Str+'无条件舍去为:'+FloatToStr(Trunc(123.56))+#10;
  ShowMessage(Str);
end;
```

- *Pi(π)*

引用单元：System
函数声明：function Pi: Extended;
返回值为：3.1415926535897932385

- *Sqr(平方)*

引用单元：System
函数声明：function Sqr(X: Extended): Extended;

- *Sqrt(平方根)*

引用单元：System
函数声明：function Sqrt(X: Extended): Extended;

- *Ln(自然对数)*

引用单元：System
函数声明：function Ln(X: Real): Real;

Exp(指数)

引用单元：System
函数声明：function Exp(X: Real): Real;

例 E-80

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S1,S: string;
  R:Real;
begin
  //将 3.9 的平方转为字符串存为字符串 S
  Str(Sqr(3.9):8:3,S);
  S1:='3.9 的平方是:'''+S+''''+#10;
  //将 3.9 开平方根转为字符串并存为字符串 S
  Str(Sqrt(3.9):8:3,S);
  S1:=S1+'3.9 的平方根是:'''+S+''''+#10;
  //求 3.9 的指数值
  Str(Exp(3.9):8:3, S);
  S1:=S1+'3.9 的指数为:'''+S+''''+#10;
  //找 3.9 的对数
  Str(ln(3.9):8:3, S);
  S1:=S1+'3.9 的指数为:'''+S+'''';
  Showmessage(S1);
end;
```

FloatToDecimal(将浮点数转为十进制数)

引用单元：SysUtils
函数声明：procedure FloatToDecimal(var DecVal: TFloatRec; const Value; ValueType: TFloatValue; Precision, Decimals: Integer);

FloatToText(将浮点数转换为指定格式的十进制数)

引用单元：SysUtils
函数声明：function FloatToText(Buffer: PChar; const Value; ValueType: TFloatValue; Format: TFloatFormat; Precision, Digits: Integer): Integer;

FloatToTextFmt(将浮点数转换为指定格式的十进制数)

引用单元：SysUtils
函数声明：function FloatToTextFmt(Buffer: PChar; const Value; ValueType: TFloatValue; Format: PChar): Integer;

- *FloatToStr(将浮点数转为字符串)*

引用单元：SysUtils
函数声明：function FloatToStr(Value: Extended): string;

- *FloatToStrF(将浮点数转为指定格式的字符串)*

引用单元：SysUtils
函数声明：function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): string;

- *FormatFloat(将浮点数转为指定格式的字符串)*

引用单元：SysUtils
函数声明：function FormatFloat(const Format: string; Value: Extended): string;

范例 E-81

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
   //FloatToStr(将浮点数转为字符串)
   Canvas.TextOut(10,10,FloatToStr(1234.5678));
   //FloatToStrF(将浮点数转为指定格式的字符串)有五种不同参数
Canvas.TextOut(10,30,FloatToStrF(1234.5678,ffGeneral,5,2));
            {1234.6}
   Canvas.TextOut(10,50,FloatToStrF(1234.5678,ffExponent,5,2));
            {1.2346E+03}
   Canvas.TextOut(10,70,FloatToStrF(1234.5678,ffFixed,5,2));
            {1234.60}
Canvas.TextOut(10,90,FloatToStrF(1234.5678,ffNumber,5,2));
            {1,234.60}
   Canvas.TextOut(10,110,FloatToStrF(1234.5678,ffCurrency,5,2));
            {NT$1,234.60}
   //FormatFloat(将浮点数转为指定格式的字符串)各种不同的格式
   Canvas.TextOut(10,130,FormatFloat('',1234.5678));  1234.5678}
   Canvas.TextOut(10,150,FormatFloat('0',1234.5678)); {1235}
   Canvas.TextOut(10,170,FormatFloat('0.00',1234.5678)); {1234.57}
   Canvas.TextOut(10,190,FormatFloat('0.00',1234.5678)); {1234.57}
   Canvas.TextOut(10,210,FormatFloat('#.##',1234.5678)); {1234.57}
   Canvas.TextOut(10,230,FormatFloat('#,##0.00',1234.5678));
             {1234.57}
   Canvas.TextOut(10,250,FormatFloat('#,##0.00;(#,##0.00)',-
```

```
                1234.5678)); {(1234.57)}
  Canvas.TextOut(10,270,FormatFloat('#,##0.00;;零',0)); {零}
Canvas.TextOut(10,290,FormatFloat('0.000E+00',1234.5678));
          {1.235E+03}
  Canvas.TextOut(10,310,FormatFloat('#.###E-0',1234.5678));
            {1.235E3}
end;
```

Random(产生随机数)

引用单元：System
函数声明：function Random [(Range: Integer)];
参数说明：如果有指定参数值，则产生的随机数介于 0 <= X < Range. 如果没有指定，则产生的随机数介于 0 <= X < 1.

Randomize(启用随机数产生器)

引用单元：System
函数声明：procedure Randomize;

范例 E-82

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  //加了随机数产生器，每次都会有不同的运行结果
  Randomize;
  //产生十个介于 0 到 1 之间的数
  for I := 1 to 10 do
    Canvas.TextOut(10,(I-1)*30,FloatToStr(Random));
end;
```

Dec(将变量递减)

引用单元：System
函数声明：procedure Dec(var X[; N: Longint]);

范例 E-83

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyInt : Integer;
  MyLong: Longint;
begin
```

```
  MyInt := 90;
  MyLong:= 90;
  Dec(MyInt);  {MyInt:=89}
  Dec(MyLong,9);{MyLong:=81}
end;
```

- *Inc(将变量递增)*

引用单元：System

函数声明：procedure Inc(var X [; N: Longint]);



E-84

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyInt : Integer;
  MyLong: Longint;
begin
  MyInt := 90;
  MyLong:= 90;
  Inc(MyInt);  {MyInt:=91}
  Inc(MyLong,9);{MyLong:=99}
end;
```

- *Odd(检查是否为奇数)*

引用单元：System

函数声明：function Odd(X: Longint): Boolean;



E-85

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   if Odd(0) then
      Showmessage('0 是奇数!!')
   else
      ShowMessage('0 不是奇数!!');
end;
```

- *IntToHex(将十进制转为十六进制)*

引用单元：SysUtils

函数声明：function IntToHex(Value:Integer;Digits:Integer):string;

函数声明：function IntToHex(Value: Int64; Digits: Integer): string; overload;

- *IntToStr(将十进制转为字符串)*

引用单元：SysUtils
函数声明：function IntToStr(Value: Integer): string; overload;
函数声明：function IntToStr(Value: Int64): string; overload;

- *Chr(获取 ASCII 码)*

引用单元：System
函数声明：function Chr(X: Byte): Char;



E-86

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Canvas.TextOut(10,10, IntToHex(99,8));
  Canvas.TextOut(10,30, IntToStr(99));
  Canvas.TextOut(10,50, Chr(99));
end;
```

E.4 *Variant 处理函数*

本小节将为您介绍 Delphi 6 所提供的 Variant 处理函数。这些函数稍后将有详细的范例说明。笔者这里将以列表的方式先说明每一个函数所代表的意义，如表 E.4 所示。

表 E.4

函数名称	单元文件	所代表的意义
VarArrayCreate	Variants	此函数可创建一个 variant array
VarArrayOf	Variants	此函数可创建一个一维 variant array
VarArrayDimCount	Variants	此函数可返回 Variant array 的维数
VarIsArray	Variants	此函数可判断 Variant 是否为 Array
VarIsEmpty	Variants	此函数可判断 Variant 是否为空
VarClear	Variants	此函数可将 Variant 加以清除
VarIsNull	Variants	此函数可判断 Variant 是否为 Null
VarType	Variants	此函数可返回 Variant 的类型
VarCopy	Variants	此函数可复制 Variant 变量
VarToDateTime	Variants	此函数可将 Variant 变量转为 TDateTime 类型

以下将以范例的方式说明每一个函数的使用方法。

- *VarArrayCreate(创建一个 variant array)*

引用单元：Variants
函数声明：function VarArrayCreate(const Bounds: array of Integer; VarType: Integer): Variant;

- *VarArrayOf(创建一个一维 variant array)*

引用单元：Variants
函数声明：function VarArrayOf(const Values: array of Variant): Variant;



E-87

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVariant: Variant;
begin
  MyVariant:=VarArrayCreate([0,3],varVariant);
  MyVariant[0] := 'Delphi is good!!';
  MyVariant[1] := Now;
  MyVariant[2] := -0.999;
  MyVariant[3] := VarArrayOf(['一','二','三','四']);
  ShowMessage(MyVariant[0]);
  ShowMessage(MyVariant[3][2]);
end;
```

- *VarArrayDimCount(返回 Variant array 的维数)*

引用单元：Variants
函数声明：function VarArrayDimCount(const A: Variant): Integer;



E-88

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVariant: Variant;
begin
  MyVariant:=VarArrayCreate([0,3],varVariant);
  ShowMessage(IntToStr(VarArrayDimCount(MyVariant)));
end;
```

- *VarIsArray(判断 Variant 是否为 Array)*

引用单元：Variants
函数声明：function VarIsArray(const V: Variant): Boolean;



E-89

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVariant: Variant;
begin
  MyVariant:=VarArrayCreate([0,3],varVariant);
  if VarIsArray(MyVariant) then
     ShowMessage('MyVariant is a Array')
  else
     ShowMessage('MyVariant is not a Array');
end;
```

- *VarIsEmpty(判断 Variant 是否为空的)*

引用单元：Variants
函数声明：function VarIsEmpty(const V: Variant): Boolean;

- *VarClear(将 Variant 初始化)*

引用单元：Variants
函数声明：procedure VarClear(var V: Variant);

- *VarIsNull(判断 Variant 是否为 Null)*

引用单元：Variants
函数声明：function VarIsNull(const V: Variant): Boolean;



E-90

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVariant: Variant;
begin
  if not VarIsNull(MyVariant) then
     begin
       MyVariant:=VarArrayCreate([0,3],varVariant);
       MyVariant[0] := 'Delphi  is good!!';
       VarClear(MyVariant);
       if VarIsEmpty(MyVariant) then
          ShowMessage('MyVariant is Empty')
       else
          ShowMessage('MyVariant is not  Empty');
     end;
end;
```

VarType(返回 Variant 的类型)

引用单元：Variants

函数声明：function VarType(const V: Variant): Integer;



E-91

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVariant: Variant;
begin
  MyVariant:=VarArrayCreate([0,4],varVariant);
  MyVariant[0] := 'Delphi  is good!!';
  MyVariant[1] := Now;
  case VarType(MyVariant[1]) of
      varEmpty    : showMessage('Empty');
      varNull     : showMessage('Null');
      varSmallint : showMessage('Smallint');
      varInteger  : showMessage('Integer');
      varSingle   : showMessage('Single');
      varDouble   : showMessage('Double');
      varCurrency : showMessage('Currency');
      varDate     : showMessage('Date');
      varOleStr   : showMessage('OleStr');
      varDispatch : showMessage('Dispatch');
      varError    : showMessage('Error');
      varBoolean  : showMessage('Boolean');
      varVariant  : showMessage('Variant');
      varUnknown  : showMessage('Unknown');
      varByte     : showMessage('Byte');
      varString   : showMessage('String');
      varTypeMask : showMessage('TypeMask');
      varArray    : showMessage('Array');
      varByRef    : showMessage('ByRef');
  end;
end;
```

VarCopy(复制 Variant 变量)

引用单元：Variants

函数声明：procedure VarCopy(var Dest: Variant; const Source: Variant);



E-92

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  MyVariant,MyVariant2: Variant;
begin
  MyVariant:=VarArrayCreate([0,3],varVariant);
  MyVariant[0] := 'Delphi is good!!';
  MyVariant[1] := Now;
  MyVariant[2] := -0.999;
  MyVariant[3] := VarArrayOf(['一','二','三','四']);
  //将 MyVariant 的内容复制到 MyVariant2
  VarCopy(MyVariant2,MyVariant);
  ShowMessage(MyVariant2[3][2]);
end;
```

VarToDateTime(将 Variant 变量转为 TDateTime 类型)

引用单元：Variants
函数声明：function VarToDateTime(const V: Variant): TDateTime);

范例 E-93

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVariant: Variant;
begin
  MyVariant:=VarArrayCreate([0,3],varVariant);
 MyVariant[0] := '99/09/24';
ShowMessage(DateTimeToStr(VarToDateTime(MyVariant[0])));
end;
```

一点通系列：



《光盘刻录一点通》
《外行学电脑一点通》
《电脑常见故障与日常维护一点通》
《BIOS与Windows注册表设置一点通》

现用现查红宝书系列：



《电脑常见故障1000例详解》
《电脑新手万用入门基础教程》
《Office XP办公自动化基础教程》
《家庭、宿舍、办公室、网吧组网实务》
《电脑系统维护与常用工具软件100款详解》
《新手上网、邮件聊天、网页制作实务全书》
《Windows9X/Me/NT/2000/XP注册表修改实务》

查询辞典系列：

《Flash Action Script语法参考辞典》
《PHP函数库查询辞典》
《C++函数库查询辞典》
《Linux指令辞典》
《五笔字型速查辞典》





完全手册系列：

《电脑硬件与维护完全手册》
《电脑软件故障处理完全手册》
《家用电脑使用与维护完全手册》

学生学电脑系列：



《网站设计与制作》
《完全接触动画制作》
《C语言程序设计初探》
《巧学活用Word文字处理》
《快速掌握Excel电子表格》
《数字图像处理应用与基础》
《轻松制作PowerPoint幻灯片》
《易学易用Authorware5.x多媒体制作》

电脑应用智囊库系列

《电脑高手修炼秘笈》



《解密X档案——电脑硬件全景秀》



《安居乐业——将“非常网络”建设到底》

